# PERANCANGAN SISTEM PENGAMANAN DATA TRANSAKSI PADA *DATABASE* TERDISTRIBUSI MENGGUNAKAN METODE *HASHING*

**SKRIPSI**

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Agung Pambudi
145150200111062

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

# PENGESAHAN

PERANCANGAN SISTEM PENGAMANAN DATA TRANSAKSI PADA *DATABASE* TERDISTRIBUSI MENGGUNAKAN METODE *HASHING*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:
Agung Pambudi
NIM: 145150200111062

Skripsi ini telah diuji dan dinyatakan lulus pada
1 Agustus 2018
Telah diperiksa dan disetujui oleh:

| Dosen Pembimbing I | Dosen Pembimbing II |
|---|---|
| Ari Kusyanti, S.T, M.Sc | Mahendra Data, S.Kom., M.Kom |
| NIK: 2011028312282001 | NIK: 2015038611171001 |

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 197105182003121001

# PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 3 Agustus 2018

Agung Pambudi

NIM: 145150200111062

# KATA PENGANTAR

Puji syukur kehadirat Allah SWT Yang Maha Esa penulis dapat menyelesaikan skripsi ini, jika bukan atas kehendaknya mustahil penulis dapat menyelesaikan skripsi untuk memperoleh gelar sarjana. Selain atas kehendak Allah SWT penulis menyadari banyak pihak yang membantu secara langsung maupun tidak langsung yang tidak dapat penulis sebutkan seluruhnya, untuk itu penulis ucapkan terimakasih sedalam-dalamnya dan semoga Allah SWT membalas kebaikan dan bantuan yang telah diberikan pada penulis. Tak lupa Penulis ingin menyampaikan terimakasih khusus kepada:

1. Orang Tua penulis atas bimbingan, do'a dan kerja kerasnya untuk membiayai kuliah penulis sehingga penulis dapat menyelesaikan skripsi untuk mendapat gelar sarjana.
2. Keluaraga penulis atas bantuan, do'a dan motifasinya.
3. Ibu Ari Kusyanti, S.T, MSc selaku pembimbing I atas bimbingan, saran dan motifasinya sehingga penulis dapat menyelesaikan skripsi ini.
4. Bapak Mahendra Data, S.Kom., M.Kom selaku pembimbing II atas bimbingan, saran dan motifasinya sehingga penulis dapat menyelesaikan skripsi ini.
5. Seluruh Dosen yang pernah membimbing dan memberi bekal ilmu pada penulis untuk menyelesaikan skripsi ini.
6. Teman-teman penulis atas bantuan, do'a dan motifasinya.

Penulis menyadari bahwa skripsi ini tidaklah sempurna untuk itu penulis membuka kesempatan bagi siapapun yang telah membaca skripsi ini untuk menyampaikan kritik dan saran demi kemajuan ilmu pengetahuan dan pengembangan penelitian selanjutnya.

Malang,3 Agustus 2018

Penulis

agungpambudi@merahputih.id

# ABSTRAK

Dalam melakukan perdagangan, toko *online* membutuhkan gerai pembayaran sebagai perantara, agar uang yang pembeli miliki dapat diteruskan ke toko *online*. Saling terhubungnya transaksi antara loket *online* kereta api dengan beberapa gerai pembayaran, menuntut adanya distribusi *database,* hal ini akan menimbulkan masalah baru, karena memiliki celah keamanan. Yaitu manipulasi data transaksi. Sehingga diperlukan sebuah sistem yang dapat menjamin data transaksi tidak dapat dimanipulasi. Menggunakan metode hashing, sistem akan menghitung nilai digest dari transaksi yang masuk dari gerai pembayaran, untuk dikirimkan secara bergiliran dengan data transaksi ke server loket online kereta api. Kemudian akan dihitung nilai digest dari dari data transaksi. Serta digest catatan transaksi sebelumnya, untuk dicatatkan pada database server loket online, agar gerai pembayaran melakukan pencatatan yang sama pada database gerai pembayaran. Dengan cara ini hasil catatan loket online kereta api, dan gerai pembayaran akan sama dan setiap catatan akan terikat. Sehingga perubahan satu catatan transaksi, akan merusak susunan catatan transaksi yang lain. Susunan catatan yang valid akan menjadi indikasi terjaganya integritas data transaksi.Kata

kunci: kriptografi, *hashing*, integritas, *database*, distribusi.

# ABSTRACT

*In trading, online stores need payment outlets as intermediaries, so that the money that the buyer has can be forwarded to an online store. The interconnection of transactions between online train counters and several payment outlets, demanding a database distribution will cause new problems, because it has a security gap, the gap is transaction data manipulation. For this reason the author makes a system that can guarantee transaction data cannot be manipulated. Using the hashing method, the system will calculate the digest value of the incoming transactions from the payment counter, to be sent in turn with transaction data to the online train ticket server. Then the digest value will be calculated from the transaction data. As well as the digest of the previous transaction records, to be recorded on the database of online counter servers, so that payment outlets do the same recording on the payment counter database. In this way the results of the train ticket online record, and the payment counter will be the same and each record will be bound. So that a change in a transaction record, will damage the composition of other transaction records. A valid record arrangement will be an indication of the integrity of transaction data.*

*Keywords*: *cryptography*, *hashing*, *Integrity*, *database*, *distribution*.

# DAFTAR ISI

# DAFTAR TABEL

# DAFTAR GAMBAR

# DAFTAR LAMPIRAN

# BAB 1 PENDAHULUAN

## 1.1 Latar Belakang

Transaksi jual beli telah dilakukan manusia sejak lama, jual beli merupakan cara untuk memenuhi kebutuhan hidup manusia. Seiring berkembangnya teknologi informasi, banyak perusahaan yang memanfaatkan teknologi untuk menjual produk atau jasa, salah satunya dengan mendirikan toko *online*. Toko *online* menyediakan berbagai macam barang dan jasa yang memudahkan konsumen untuk memilih barang atau jasa dan juga untuk melakukan pembayarannya, banyak pilihan metode pembayaran dalam berbelanja *online* tergantung pada kebijakan toko *online* masing-masing. Sebagian besar pembayaran *online* tidak ditangani langsung oleh toko *online*, namun dipercayakan pada beberapa gerai pembayaran yang telah menjadi mitra toko *online* tersebut. Hal ini menimbulkan masalah baru karena data transaksi tidak dibuat oleh toko *online*. Data transaksi dibuat oleh gerai pembayaran sehingga toko *online* hanya mendapat salinan data transaksi dari gerai mitranya, maka dari itu diperlukan sistem pengamanan data transaksi. Agar data yang telah dibuat oleh gerai pembayaran terjaga integritasnya dan tidak merugikan toko *online*. Mengingat fungsi data transaksi adalah sebagai bukti yang sah, sehingga data transaksi harus dapat dipertanggungjawabkan.

Fungsi *hash* memiliki peran mendasar dalam kriptografi modern, fungsi *hash* digunakan untuk integritas data bersama dengan skema tanda tangan digital, karena beberapa tujuan sebuah pesan biasanya di-*hash* terlebih dahulu, ide dasar dari fungsi *hash* kriptografi adalah nilai *hash* berfungsi sebagai gambaran representatif yang ringkas (terkadang disebut jejak, sidik jari digital, atau pesan yang olah) dari masukan *string*, sehingga kombinasi karakter unik dapat diidentifikasi sebagai *string* tersebut (Menezes, dkk., 1996).

Pada penelitian sebelumnya yang dilakukan oleh Satoshi Nakamoto, berjudul "*Bitcoin: A Peer-to-Peer Electronic Cash System*", mengulas bagaimana seorang Satoshi Nakamoto mengamankan sebuah data transaksi perdagangan dengan mata uang *Bitcoin,* menggunakan skema yang diberi nama *timestamp*. Maksud dari timestamp ialah, setiap stempel waktu menyertakan stempel waktu sebelumnya dalam *hash* yang baru, membentuk rantai, dengan masing-masing stempel waktu memperkuat yang sebelumnya(Nakamoto, 2008). Hal ini yang menjadi isnpirasi penulis, untuk menyalesaikan masalah yang sebelumnya telah disampaikan. Menggunakan skema yang sama, data yang masuk di *server* gerai dilakukan penghitungan *hash*, dengan disertakan *hash* catatan sebelumnya. Sehingga catatan pada *database* membentuk sebuah rantai, satu catatan akan memperkuat cacatan lainnya. Sehingga catatan data transaksi yang dikirimkan ke server toko *online*, akan terjamin itegritasnya. Begitu juga dengan server toko *online*, akan menerapkan skema yang sama agar integritas catatannya tetap terjaga. Jika Satoshi Nakamoto menerapkan skema tersebut pada *Bitcoin*, dalam penelitian ini penulis akan mengambil studi kasus toko *online* penjual tiket kereta api, agar lebih mudah dalam melakukan pemahaman.

## 1.2 Rumusan masalah

1. Algoritma *hashing* apa yang digunkan untuk mengamankan transaksi pada *database* terdistribusi ?
1. Bagaimana cara melakukan implementasi algoritme *hashing* ke dalam sistem pengamanan data transaksi ?
2. Bagaimana cara mengamankan data transaksi menggunakan metode *hashing* agar tidak dapat di manipulasi?

## 1.3 Tujuan

1. Memastikan algoritme *hashing* yang akan digunkan untuk mengamankan transaksi pada *database* terdistribusi telah sesuai.
2. Melakukan implementasi algoritme *hashing* ke dalam sistem pengamanan data transaksi.
3. Mengamankan data transaksi menggunakan metode *hashing* agar tidak dapat dimanipulasi.

## 1.4 Manfaat

1. Integritas data transaksi dapat dijaga dengan algoritme yang sesuai.
2. Sistem telah terintegrasi dengan algoritme *hashing* dan dapat berjalan susuai fungsinya.
3. Data transaksi tidak dapat dimanipulasi oleh gerai-gerai resmi dan gerai-gerai mitra maupun pihak lain.

## 1.5 Batasan masalah

1. Karena sifat sistem perusahaan yang begitu kompleks, dan penulis hanya berfokus pada pengamanan data transaksi. penulis hanya merancang dan melakukan simulasi pertukaran data transaksi tanpa membuat sistem perusahaan secara penuh.
2. Sistem transaksi membutuhkan data hasil pemesanan, karena tidak adanya sistem pemesanan penulis akan mengisikan data pemesanan secara manual kedalam *database* pemesanan.
3. Penulis hanya memberikan gambaran jika sistem ini diterapkan pada kasus penjualan tiket kereta api namun tidak untuk menggantikan sistem yang saat ini digunakan sehingga penulis tidak melakukan penelitian bagaimana sistem penjualan tiket PT. Kereta Api sebenarnya bekerja.

## 1.6 Sistematika pembahasan

Untuk lebih mudah memahami isi dari skripsi ini, maka isi yang tertera pada skripsi ini dikelompokkan kedalam beberapa bab dengan sistematika penulisan sebagai berikut:

**Bab I: Pendahuluan**

Berisi tentang latar belakang diangkatnya masalah data transaksi ini kedalam sebuah karya tulis ilmiah, perumusan masalah, tujuan dan manfaat dari penelitian ini, ruang lingkup penelitian, dan sistematika penulisan.

**Bab II: Landasan Teori**

Bab ini berisikan teori yang berupa pengertian dan definisi yang diambil dari publikasi mengenai *SHA-1* beserta penjelasannya dan teori tentang kriptografi guna menunjang penelitian ini.

**Bab III: Metodologi**

Bab ini berisikan tentang langkah-langkah yang akan dilakukan selama penelitian untuk dapat menyelesaikan masalah yang diangkat, dalam hal ini untuk merancang sistem pengamanan data transaksi yang diharapkan dapat dilakukan implementasi.

**Bab IV: Perancangan**

Bab ini berisi penjelasan bagaimana penulis melakukan perancangan sistem pengamanan data transaksi dan penjelasan bagaimana data transaksi akan diamankan.

**Bab V: Implementasi**

Bab ini berisi bagaimana rancangan yang telah penulis buat sebelumnya akan dilakukan implementasi kedalam kode program hingga sistem dapat dijalankan sebagaimana mestinya.

**Bab VI: Pengujian**

Bab ini berisikan bagaimana sistem pengamanan data transaksi pada database terdistribusi yang sebelumnya telah dirancang dan dilakukan implementasi, akan diuji apakah penerapan metode *hashing* pada sistem ini dapat mengatasi masalah yang telah disebutkan sebelumnya.

**Bab VIII: Kesimpulan**

Bab ini berisi kesimpulan dan saran yang penulis peroleh setelah melakukan penelitian berkaitan dengan penerapan metode *hashing* pada sistem pengamanan data transaksi, sebagaimana yang telah diuraikan pada bab-bab sebelumnya.

# BAB 2 LANDASAN KEPUSTAKAAN

## 2.1 Kajian Pustaka

Kemajuan teknologi adalah buah dari penelitian orang-orang terdahulu, satu penelitian akan menunjang penelitian lain dikemudian hari. Begitu juga penulis dalam melakukan penelitian ini ditunjang penelitian yang telah dipublikasikan sebagai berikut.

| No | Nama Peneliti, Tahun, dan Judul | Persamaan | Perbedaan | |
|---|---|---|---|---|
| | | | Penelitian sebelumnya | Penelitian penulis |
| 1 | Satoshi Nakamoto, (2008), " Bitcoin: A Peer-to-Peer Electronic Cash System | Menggunakan algoritme *Secure Hash Algorithm* | Implementasi pada Bitcoin dengan menggunkan block | Implementasi pada sistem pengamanan data transaksi dengan menggunakan *database* |
| 2 | Angger Ardyanto Pamungkas, (2006),” IMPLEMENTASI ALGORITMA MD5, SHA1, DAN RC4 UNTUK SISTEM KRIPTOGRAFI PADA APLIKASI MOBILE INTERNET BERBASIS JAVA” | Menggunakan algoritme *Secure Hash Algorithm* 1 | Implementasi pada aplikasi *mobile* internet berbasis java | Implementasi pada sistem pengamanan data transaksi |
| 3 | Revati Raman Dewanagn, , Deepali Thombre, Kaushlendra Sharma (2015), " Implementation of Secure Hash Algorithm Using JAVA Programming" | Menggunakan algoritme *Secure Hash Algorithm 1* | Implementasi dengan menggunakan bahasa pemrograman JAVA | Implementasi dengan menggunakan bahasa pemrograman Python |

**Tabel 2.1 Kajian Pustaka**

## 2.2 Dasar Teori

Dalam melakukan penelitian ini penulis mencoba menyelesaikan masalah yang ada dengan menggunakan metode dan algoritme sebagai berikut.

### 2.2.1 Kriptografi

Kriptografi adalah sebuah bidang studi matematika yang berkaitan tentang teknik pengamanan informasi dengan menggunakan rumus matematika yang telah dirancang sedemikian rupa agar dapat menjamin keamanan informasi. Kriptografi sendiri memiliki arti tulisan rahasia, diambil dari bahasa yunani yaitu *cryptós* (rahasia) dan *gráphein* (tulisan). Kriptografi memiliki sejarah yang panjang, sekitar 4000 tahun yang lalu orang-orang mesir kuno telah memakai teknik kriptografi untuk mengamankan informasi, kriptografi juga populer pada perang dunia kedua dan memegang peran penting bagi hasil peperangan, selain untuk fungsi militer kriptografi juga digunakan untuk layanan diplomatik dan pemerintahan suatu negara hingga saat ini, tentunya dengan teknik pengamanan yang lebih dikembangkan seiring berjalannya waktu. Pada awalnya kriptografi bekerja dengan cara menyandikan pesan asli yang dapat dimengerti kedalam bentuk yang tidak lagi dapat dimengerti, hal ini untuk menunjang komunikasi antar suatu golongan agar tidak dimengerti oleh golongan lain terlebih oleh musuh, namun dewasa ini kriptografi tidak lagi digunakan untuk sekedar menjaga kerahasiaan (*Confidentiality*) namun juga untuk menjaga integritas data (*Integrity*), Autentik (*Authentication*) (Menezes, dkk., 1996).

#### 2.2.1.1 Kerahasiaan (*Confidentialiy*)

Kerahasiaan adalah tujuan kriptografi yang pertama, fungsinya untuk melindungi isi dari sebuah informasi agar tidak diketahui oleh orang yang tidak memiliki hak atau kewenangan. Untuk menjaga kerahasiaan dengan kriptografi ialah menggunakan metode enkripsi, metode enkripsi bekerja dengan cara mengubah isi dari sebuah informasi menggunakan perhitungan matematika serta menyisipkan sebuah kunci agar tidak lagi bisa dibaca, jika ingin membuka informasi tersebut haruslah mengetahui kunci yang telah diberikan dan melakukan perhitungan matematika agar informasi tersebut dapat dibaca kembali atau disebut dengan dekripsi. Pesan asli sebelum diubah disebut sebagai *plaintext* sedangkan pesan yangtelah diubah dan tidak dapat dibaca disebut sebagai *ciphertext*. Kekautan sebuah enkripsi terletak pada algoritme, algoritme menentukan bagaimana enkripsi bekerja serta metode untuk mengubah sebuah *plaintext* menjadi sebuah *ciphertext*, semakin baik algoritme maka semakin terjamin informasi yang dilindungi serta semakin susah untuk diretas. Enkripsi menurut kuncinya dibagi dua, yang pertama ialah enkripsi kunci simetris dan kunci yang digunakan untuk enkripsi dan dekripsi sama kemudian jenis yang kedua ialah kunci asimetris dan kunci yang digunakan untuk enkripsi adalah kunci publik dan kunci yang digunkan untuk dekripsi adalah kunci privat. Enkripsi tidak hanya dibedakan menurut kuncinya melainkan enkripsi juga dibedakan menurut metode

perhitungannya, yang pertama ialah enkripsi blok *Cipher* dan bentuk biner dari sebuah informasi akan dikelompokkan menjadi beberapa blok untuk kemudian dilakukan perhitungan perblok dan yang kedua ialah Stream *Cipher* dan bentuk biner dari sebuah informasi akan dihitung satu persatu (Menezes, dkk., 1996).

#### 2.2.1.2 Integritas (*Integrity*)

Integritas data adalah tujuan kriptografi yang kedua, fungsinya untuk mencegah terjadinya perubahan data yang tidak sah. Untuk menjamin integritas data, seseorang harus memiliki kemampuan untuk mendeteksi manipulasi data oleh pihak yang tidak berwenang. Manipulasi data meliputi hal-hal seperti penyisipan, penghapusan, dan substitusi. Metode kriptografi untuk mendeteksi manipulasi data ialah *hashing*. *Hashing* bekerja dengan cara mengubah sebuah pesan atau dalam kriptografi disebut *message* menggunakan perhitungan matematika menjadi beberapa karakter tidak bisa dibaca yang mewakili nilai dari sebuah pesan yang disebut sebagai *digest*, *digest* tidak bisa diubah kembali menjadi *message* dan nilai *digest* akan berbeda pada tiap-tiap kombinasi pesan namun panjang *digest* akan selalu sama, hal ini berguna untuk membandingkan nilai *digest* pesan asli dengan pesan yang ingin diuji keasliannya, jika nilai *digest* sama berarti pesan tersebut asli dan tidak dimanipulasi namun jika nilai *digest* berbeda berarti pesan tersebut telah dimanipulsi, entah itu diubah atau disisipi sesuatu (Menezes, dkk., 1996).

#### 2.2.1.3 Autentikasi (*Authentication*)

Autentikasi adalah hal yang terkait dengan identifikasi. Fungsi ini berlaku untuk entitas dan informasi itu sendiri. Dua pihak yang akan berkomunikasi harus saling mengidentifikasi. Informasi yang disampaikan melalui jaringan harus diperiksa autentiknya seperti asal, tanggal asal, isi data, waktu dikirim, dan sebagainya, tujuannya untuk memastikan bahwa sesorang atau sesuatu adalah autentik atau asli. Autentik dibagi menjadi dua: autentik terhadap seseorang dan autentik terhadap objek. Autentik terhadap objek dilakukan untuk memastikan objek tersebut autentik dan tidak diubah oleh orang lain sementara autentik terhadap seseorang adalah memastikan pengirim data adalah orang yang benar dan tidak mengatasnamakan orang lain, autentik ini dapat menggunakan metode Digital Signature. Digital Signature bekerja dengan paduan antara *hashing* dan enkripsi, pertama-tama pengirim data akan melakukan penandatanganan atau signing pada dokumen sebelum dokumen tersebut dikirim, setelah dokumen dikirim penerima harus melakukan verifikasi atau verify untuk memastikan dokumen tersebut autentik (Menezes, dkk., 1996).

### 2.2.2 *Secure Hash Algorithm*

*Secure Hash Algorithm* (*SHA*) adalah sebuah algoritme *hashing* yang dapat mengubah pesan (*Message*) menjadi karakter tidak dapat dibaca yang merepresentasikan inti sari dari sebuah pesan (*Digest*), algoritme ini dirancang oleh National Institute *of Standard* and Technology (*NIST*) dan mulai

diperkenalkan pada bulan agustus tahun 1991 di Amerika Serikat dan merupakan bagian dari Digital Signature *Standard* (DSS) yang dijadikan standar pemrosesan informasi pemerintah Amerika Serikat atau disebut U.S. Federal Information Processing *Standard* (*FIPS*). Beberapa versi *Secure Hash Algorithm* tergabung dalam *secure hash Standard* (*SHS*) (Dang, 2015). Dikembangkan untuk menggantikan algoritme-algoritme *hashing* sebelumnya yang dinilai telah tidak aman lagi untuk digunakan di lingkungan pemerintahan Amerika Serikat. *Secure Hash Algorithm* terus digunakan sampai sekarang dengan beberapa kali pembaruan dan saat ini versi yang paling baru ialah algoritme Keccak yang memenangkan sayembara dan berhak menyandang nama *Secure Hash Algorithm* 3 (*SHA*-3) serta resmi menggantikan *Secure hash Algorithm* 2 (*SHA*-2) sebagai standar algoritme *hashing.*

#### 2.2.2.1 *Secure Hash Algorithm* 1 (*SHA-1*)

*SHA-1* adalah salah satu varian dari keluarga *Secure Hash Algorithm* (*SHA*) yang merupakan bagian dari *Secure Hash Standard* (*SHS*) dengan fungsi yang sama yaitu mengubah pesan atau *message* (M) menjadi sebuah *digest*, M memiliki panjang sebanyak l bits($0 \leq l < 2^{64}$), algoritme ini menggunakan 1) penjadwalan pesan dari delapan puluh potongan pesan masing-masing 32 bit 2) lima variabel masing-masing 32 bit, dan 3) lima potongan *hash* masing-masing 32 bit. Sehingga akan *SHA-1* akan menghasilkan 160 bit *digest*. Potongan pesan pada saat penjadwalan akan diberi label W0, W1 sampai W79. Lima variabel kerja masing-masing akan diberi label a, b, c, d, dan e. potongan nilai *hash* akan diberi label H (i ) , H (i ) , … , H (i ) , nilai H akan diproses sendiri-sendiri dan akan disatukan kembali menjadi sebuah *digest*. *SHA-1* hanya menggunakan satu penyimpanan sementara yaitu TEMP (T) (Dang, 2015).

#### 2.2.2.2 Message Padding *SHA-1*

*SHA-1* menggunakan blok dengan besaran 512 bit, tidak ada jumlah minimum atau maksimum yang membatasi jumlah pesan masukan, berapapun jumlah masukan akan tetap diproses dan menghasilkan *digest* dengan besaran 160 bit. Karena jumlah blok untuk pemrosesan adalah 512 bit dan masukannya belum tentu 512 bit, digunakan *message padding* yang akan menambah besaran bit pesan masukan menjadi kelipatan 512 agar dapat diproses oleh *SHA-1*. Cara kerja *padding* ialah menambahkan karakter “1” yang diikuti oleh karakter “0” sebanyak m buah serta 64 bit *integer* pesan asli sebelum dilakukan *padding* pada akhiran pesan untuk menghasilkan pesan dengan panjang 512 bit (Dang, 2015).

#### 2.2.2.3 Komputasi *SHA-1*

Berikut adalah fungsi untuk menghitung *SHA-1* dan setiap blok pesan, M(1), M(2), …, M(N), diproses secara berurutan dengan langkah sebagai berikut:

Pertama-tama sebelum melakukan proses komputasi H diinisialisasi sebagai berikut:

$H_0$ = 67452301

$H_1$ = EFCDAB89

$H_2$ = 98BADCFE

$H_3$ =10325476

$H_4$ = C3D2E1F0

Kemudian inisialisasi fungsi (f) pada persamaan (2.1) dan Konstanta (k) pada persamaan (2.2) sebagai berikut:

$$f_t(x,y,z) = \begin{cases} Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z) & 0 \le t \le 19 \\ Parity(x,y,z) = x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x,y,z) = x \oplus y \oplus z & 60 \le t \le 79. \end{cases} \quad (2.1)$$

$$K_t = \begin{cases} 5a827999 & 0 \le t \le 19 \\ 6ed9eba1 & 20 \le t \le 39 \\ 8f1bbcdc & 40 \le t \le 59 \\ ca62c1d6 & 60 \le t \le 79. \end{cases} \quad (2.2)$$

Berikutnya lakukan komputasi:

*For* i=1 to N:

{

1. Mempersiapkan penjadwalan pesan sesuai persamaan (2.3) {$W_t$}:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ ROTL^1 (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \le t \le 79 \end{cases} \quad (2.3)$$

2. Inisialisasi lima variabel a, b, c, d, dan e, dengan nilai *hash* (i-1):

$a = H_0^{(i-1)}$

$b = H_1^{(i-1)}$

$c = H_2^{(i-1)}$

$d = H_3^{(i-1)}$

$e = H_4^{(i-1)}$

3. *For* t=0 to 79:

$T = ROTL^5(a) + f_t(b,c,d) + e + K_t + W_t$

$e = d$

$d = c$

$c = ROTL^{30}(b)$

$b = a$

$a = T$

4. Menghitung nilai $H^{(i)}$:

$H_0^{(i)} = a + H_0^{(i-1)}$

$H_1^{(i)} = b + H_1^{(i-1)}$

$H_2^{(i)} = c + H_2^{(i-1)}$

$H_3^{(i)} = d + H_3^{(i-1)}$

$H_4^{(i)} = e + H_4^{(i-1)}$

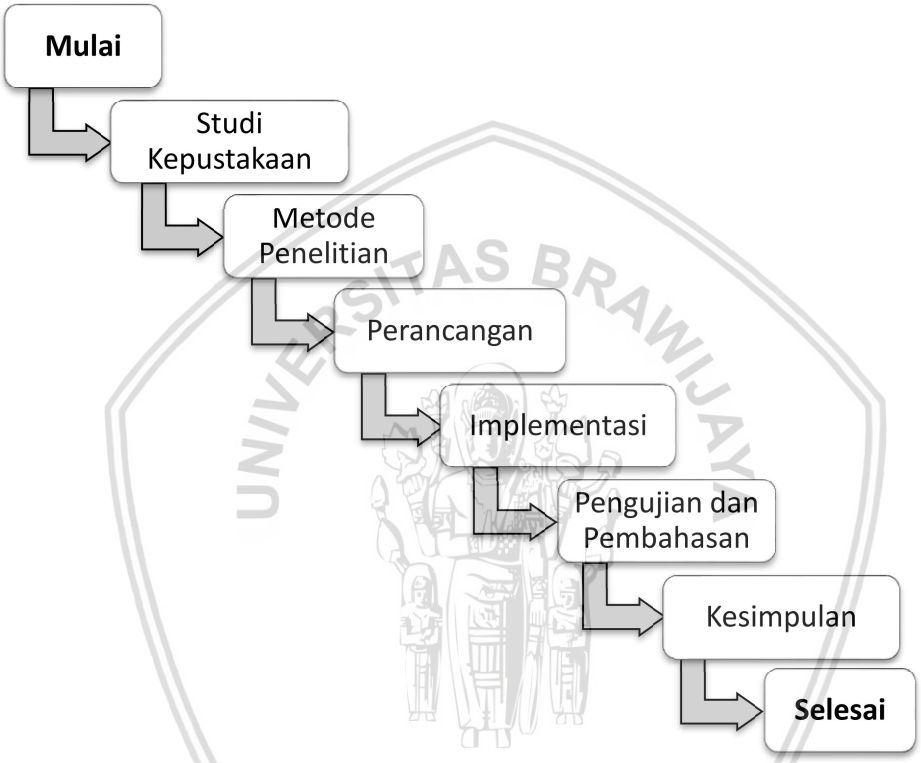
}

Setelah melakukan perhitungan empat langkah diatas akan menghasilkan 160-bit *digest* dengan cara menggabungkan nilai H0 sampai H4.

$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)}$

# BAB 3 METODOLOGI

Bab Metodologi ini akan menjelaskan langkah-langkah yang dilakukan penulis selama penelitian, langkah-langkah dilakukan berurutan antar tahapnya dengan judul penelitian "Perancangan Sistem Pengamanan Data Transaksi Pada *Database* Terdistribusi Menggunakan Metode *Hashing*". Berikut adalah tahapan yang dilakukan penulis dalam melakukan penelitian dan dapat dilihat pada Gambar 3.1.




**Gambar 3.1 Langkah Penelitian**

## 3.1 Studi Kepustakaan

Penelitian-penelitian yang telah dilakukan saat ini adalah buah dari penelitian yang telah lampau, melakukan penelitian saat ini akan menghasilkan penelitian baru dimasa depan baik oleh diri sendiri maupun orang lain, melakukan penelitian membutuhkan pemahaman yang mendalam terhadap penelitian yang relevan agar hasil penelitian dapat dipertanggung jawabkan kebenarannya, maka dari itu penulis melakukan studi kepustakaan terhadap penelitian yang telah ada yang relevan dengan penelitian penulis saat ini yaitu tentang Kriptografi, tujuan Kriptografi dan *Secure Hash Standard* yang penulis peroleh dari Buku dan publikasi ilmiah.

## 3.2 Metode Penelitian

Pada metode penelitian ini penulis menjelaskan tahapan-tahapan yang akan penulis lakukan selama penulis melakukan penelitian, tahapan-tahapan ini akan dilakukan setiap langkah secara berurutan agar penelitian menjadi teratur dan sistematis serta dapat menyelesaikan tujuan penelitian secara baik dan benar.

## 3.3 Perancangan

Dalam penelitian ini perancangan dimaksudkan untuk mencari rekayasa sistem yang paling sesuai agar kedepannya sistem yang telah dirancang dapat menyelesaikan masalah yang ada, sistem yang dirancang ialah sistem pengamanan data transaksi pada *database* terdistribusi yang menggunakan algoritme *hashing* sebagai metode pengamanannya. Penelitian ini tergolong penelitian implementatif, tujuan dari perancangan ini ialah mendapat model sistem yang paling sesuai untuk di implementasikan. Penulis akan melakukan penjelasan tentang kebutuhan sistem, alur kerja sistem dan skenario sistem.

## 3.4 Implementasi

Implementasi dilakukan dengan menerapkan rancangan yang sebelumnya telah dibuat kedalam kode program sesuai dengan model di perancangan hingga sistem siap untuk dijalankan, ada dua tahap implementasi, tahap pertama yaitu melakukan implementasi algoritme *Secure Hash Algorithm* 1 (*SHA-1*) kedalam kode program menggunakan bahasa pemrograman Python kemudian tahap kedua melakukan implementasi kode program *hashing* program tadi untuk menghasilkan *digest* dari data transaksi kemudian di masukan kedalam sistem *database* terdistribusi yang juga menggunakan bahasa pemrograman Python.

## 3.5 Pengujian Dan Pembahasan

Setelah sistem dirancang, dilakukan implementasi dan siap dijalankan perlu dilakukan untuk melihat apakah sistem dapat berjalan dengan baik dan juga menjadi bahan pembahasan apakah sistem dapat menyelesaikan masalah atau tidak. Pengujian dilakukan dengan terlebih dahulu membuat skenario uji untuk melakukan percobaan terhadap beberapa parameter yaitu:

1. Parameter pertama ialah akurasi hasil perhitungan *digest*, percobaan yang dilakukan untuk menguji parameter ini ialah dengan membandingkan *digest* yang dihasilkan dengan *test vector Secure Hash Algorithm* 1 (*SHA-1*) atau hasil *digest* program *Secure Hash Algorithm* 1 (*SHA-1*) lain yang sudah teruji untuk melakukan validasi bahwa algoritme *Secure Hash Algorithm* 1 (*SHA-1*) yang diterapkan telah sesuai dan dapat dipertanggung jawabkan.

2. Parameter kedua ialah kegunaan atau usability, percobaan dilakukan untuk mengetahui apakah sistem dapat bekerja dengan baik dan benar serta mengatasi masalah yang ada tanpa kendala dengan cara melakukan simulasi mulai dari sistem berjalan, data transaksi diubah menjadi *digest*, data dikirim kegerai yang lain hingga data disimpan kedalam *database* masing-masing.

## 3.6 Kesimpulan

Setelah semua tahap dilakukan dan dinilai sistem telah berjalan dengan baik dan benar, ditariklah kesimpulan yang didapat dari tahap awal hingga pengujian sehingga penulis dapat mengambil hikmah dari penelitian ini serta menuliskan saran yang bisa dijadikan rujukan bagi pembaca yang berniat melanjutkan penelitian ini agar nilai keilmuan dan pembaruan dapat berkembang.

# BAB 4 PERANCANGAN

Dalam membangun suatu sistem komputer diperlukan sebuah rancangan sebagai acuan untuk melakukan implementasi sebuah kode program menjadi sistem utuh, dalam penelitian ini penulis melakukan sebuah perancangan untuk menghasilkan rancangan yang nantinya akan menjadi acuan implementasi kode program. Pada penelitian ini ada dua fokus perancangan yang dibuat oleh penulis yaitu merancang bagaimana sistem secara keseluruhan bekerja dan merancang bagaimana bagian-bagian sistem bekerja.

## 4.1 Perancangan Algoritme

Perancangan algoritme dimaksudkan untuk mengenal lebih jauh algoritme yang akan diterapkan dalam penelitian ini,dalam perancangan algoritme ini penulis akan melakukan perhitungan manual bagaimana sebuah pesan akan diubah menjadi *digest* sesuai aturan algoritme *Secure Hash Algorithm* 1 (*SHA-1*) yang nantinya hasil perhitungan manual ini dapat dipakai sebagai *test vector* untuk pengujian nantinya, selain perhitungan manual penulis juga akan membuat digram alir algoritme untuk dilakukan implementasi kedalam bahasa pemrograman.

### 4.1.1 Cara Kerja Algoritme

Untuk mengetahui cara kerja algoritme penulis akan melakukan perhitungan manual untuk mengubah pesan menjadi *digest* dengan menggunakan algoritme *Secure Hash Algorithm* 1 (*SHA-1*), pesan yang akan dirubah menjadi *digest* ialah "agung pambudi" berikut adalah langkah-langkah perhitungan *digest*:

1. Rubah pesan "agung pambudi" kedalam bentuk biner.

   01100001011001110111010101101110011001110010000001110000011
   00001011011010110001001110101011001000110100110000000000000

2. Tambahkan "1" diakhir bilangan biner.

   01100001011001110111010101101110011001110010000001110000011
   00001011011010110001001110101011001000110100110

3. Tambahkan "0" sampai panjang bilangan sama dengan 448 *mod* 512.

   01100001011001110111010101101110011001110010000001110000011
   00001011011010110001001110101011001000110100110000000000000
   00000000000000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000000000000000000000000000
   00000000000000000000000000000000

4. Tambahkan bilangan biner asli dengan format 64 bit.

   01100001011001110111010101101110011001110010000001110000011
   00001011011010110001001110101011001000110100110000000000000
   00000000000000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000001101000

5. Bagi bilangan menjadi W sebanyak 80 bagian berformat *integer* dengan ketentuan bagian 17 sampai 80 dihitung dengan rumus ROTL1 ( Wt–3 ⊕Wt–8 ⊕Wt–14 ⊕Wt–16 ), ⊕ adalah XOR, ROTL ialah menukar n bilangan paling akhir ke awal kumpulan bilangan.

   W1 = 1634170222

   W2 = 1730179169

   W3 = 1835169124

   W4 = 1769996288

   W5 = 0

   W6 = 0

   W7 = 0

   W8 = 0

   W9 = 0

   W10 = 0

   W11 = 0

   W12 = 0

   W13 = 0

   W14 = 0

   W15 = 0

   W16 = 104

   W17 = 403308564

   W18 = 490791106

   W19 = 3670338072

   W20 = 3809738792

   W21 = 981582212

W22 = 3045708849

W23 = 3324510289

W24 = 1963164632

W25 = 1527228491

W26 = 3067199783

W27 = 1603195777

W28 = 1881624775

W29 = 413139271

W30 = 3557691313

W31 = 1544200453

W32 = 3787900010

W33 = 2609814508

W34 = 689897872

W35 = 3155851502

W36 = 2053652069

W37 = 2586673669

W38 = 1362903405

W39 = 1989832437

W40 = 1886521633

W41 = 2625061527

W42 = 854064395

W43 = 378357267

W44 = 2221688584

W45 = 3636575897

W46 = 3841169738

W47 = 1793336872

W48 = 3235177604

W49 = 3189797310

W50 = 384420268

W51 = 3772559609

W52 = 573608828

W53 = 1161770890

W54 = 1254278142

W55 = 1148400237

W56 = 2398899785

W57 = 4231097736

W58 = 3357705093

W59 = 1087900789

W60 = 2101100909

W61 = 2121292156

W62 = 1551590538

W63 = 3675670829

W64 = 1298297914

W65 = 4251816075

W66 = 1339760880

W67 = 1353973369

W68 = 3511091913

W69 = 1640954070

W70 = 2448174729

W71 = 1680222211

W72 = 3571745344

W73 = 2698100735

W74 = 1036489783

W75 = 1979601505

W76 = 2711826850

W77 = 4074699105

W78 = 3927365809

W79 = 3355279375

W80 = 1225154518

6. Inisiasi nilai a,b,c,d dan e dengan ketentuan *SHA-1*

   a = 67452301

   b = EFCDAB89

   c = 98BADCFE

   d = 10325476

e = C3D2E1F0

7. Masukan nilai t, a, b, c, d dan e kedalam rumus berikut lalu ulangi perhitungan sebanyak 80 kali, f adalah fungsi *SHA-1*, K adalah konstanta *SHA-1* dan ROTL adalah menukar n bilangan paling akhir ke awal kumpulan bilangan, perhitungan dilakukan dengan format biner 32 bit.

   $T = ROTL^5 (a) + f_t (b,c,d) + e + K_t + t_t$

   e = d

   d = c

   $c = ROTL^{30} (b)$

   b = a

   a = T

8. Hasil dari perhitungan langkah 7 dalam format *integer*.

   a = 2291184888

   b = 3428386901

   c = 4143244229

   d = 126187710

   e = 4202835696

9. Kemudian hitung dalam fomat biner 32 bit nilai H dengan rumus sebagai berikut:

   $H_0^{(i)} = a + H_0^{(i-1)}$

   $H_1^{(i)} = b + H_1^{(i-1)}$

   $H_2^{(i)} = c + H_2^{(i-1)}$

   $H_3^{(i)} = d + H_3^{(i-1)}$

   $H_4^{(i)} = e + H_4^{(i-1)}$

10. Hasil dari perhitungan langkah 9 dalam format *integer* lalu dirubah ke format. *hexadecimal.*

    $H_0^{(i)}$ = 4023769081 = efd5d7f9

    $H_1^{(i)}$ = 3156653022 = bc26b3de

    $H_2^{(i)}$ = 2410660035 = 8fafc0c3

    $H_3^{(i)}$ = 397921588 = 17b7cd34

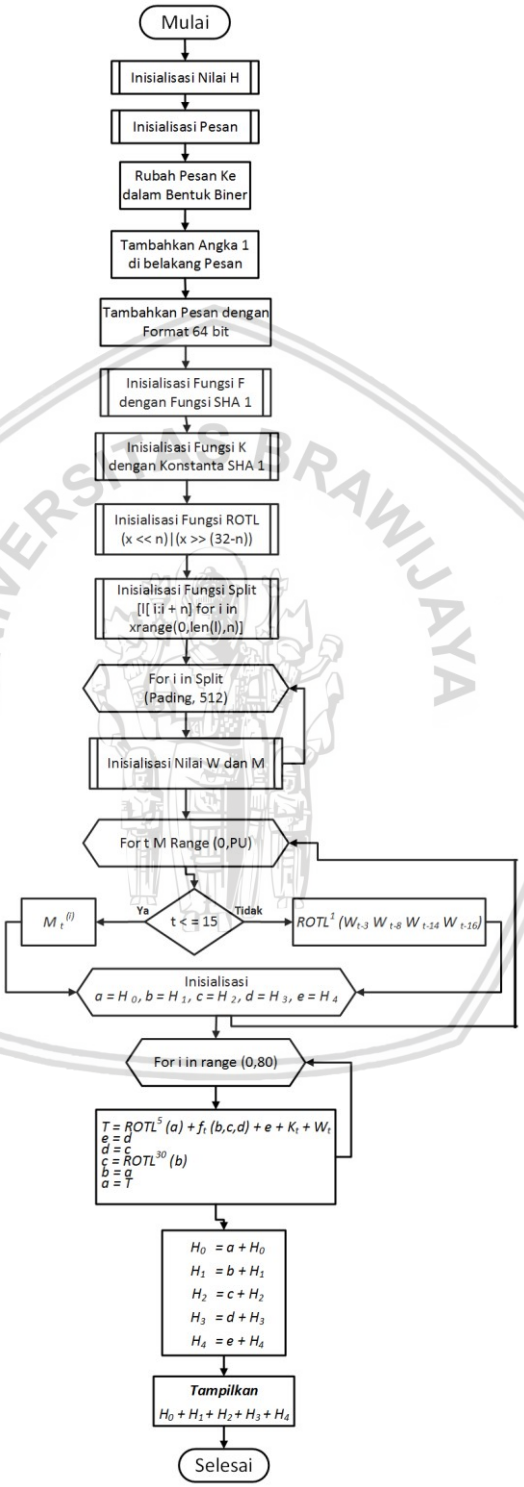
    $H_4^{(i)}$ = 3193245920 = be5510e0

11. *Digest* pesan “agung pambudi” ialah penggabungan dari nilai H dalam format *hexadecimal* sebagai berikut:

    efd5d7f9bc26b3de8fafc0c317b7cd34be5510e0

### 4.1.2 Diagram Alir Algoritme

Pada program ini penulis melakukan implementasi algoritme *SHA-1* dengan bahasa pemrograman Python, program ini digunakan sebagai modul penghitung *digest* yang akan digunakan *server* kereta dan *server* gerai.




**Gambar 4.1 Diagram alir Modul *SHA-1***

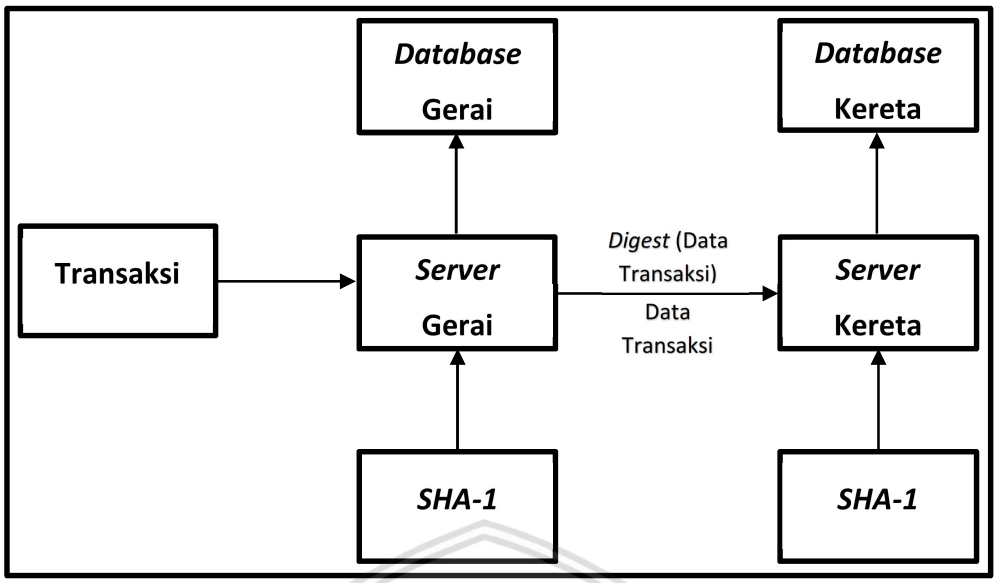
## 4.2 Perancangan Sistem

Perancangan sistem dimaksudkan untuk merancang alur kerja sistem untuk menentukan bagian-bagian yang nantinya akan membangun sistem dan akan dilakukan implementasi dengan kode program, penulis akan melakukan simulasi alur kerja sistem menggambar beberapa diagram yang dapat menjadi gambaran cara sistem bekerja, dengan dibuatnya perancangan ini diharapkan ketika melakukan implementasi tidak ada masalah yang timbul akibat ketidak jelasan alur kerja sistem.

### 4.2.1 Cara Kerja Sistem

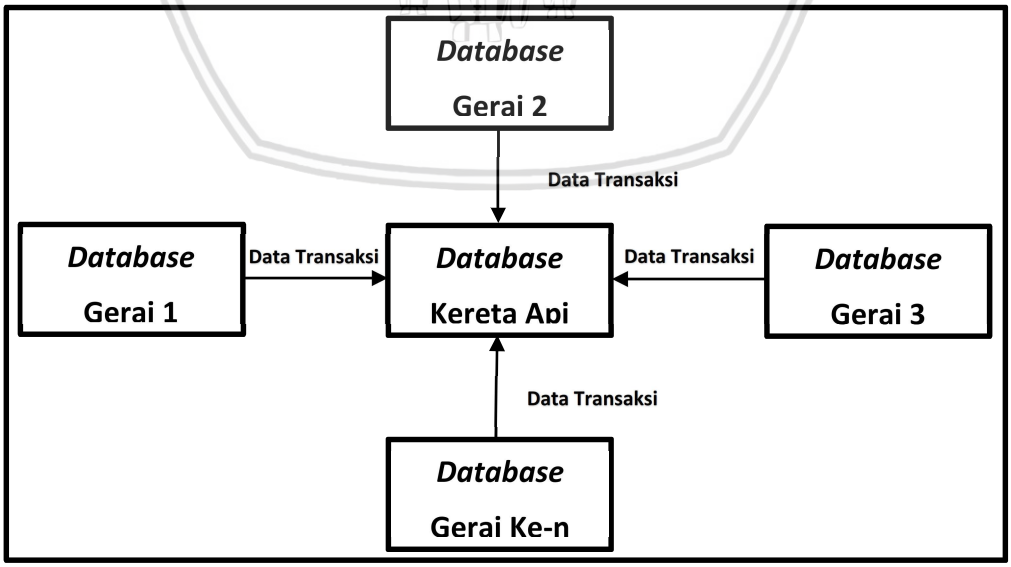
Tujuan dari Sistem Pengamanan Data Transaksi Pada *Database* Terdistribusi ialah untuk menjaga integritas data transaksi mengingat penjualan tiket kereta api saat ini didominasi dari gerai-gerai mitra secara *online* bukan lagi hanya di loket-loket setiap stasiun, gerai-gerai mitra yang jumlahnya tidak sedikit mutlak memerlukan distribusi *database* dari *database* kereta api dengan gerai-gerai lainya yang menyebabkan duplikasi data transaksi. Hal ini dapat memunculkan masalah baru jika data transaksi tidak dijaga integritasnya mengingat data transaksi memiliki peran sebagai bukti transaksi penjualan tiket kereta, jika salah satu atau beberapa gerai diretas dan dirubah catatan data transaksinya sehingga data tersebut berbeda dengan data server kereta api atau sebaliknya, bukan hanya data transaksi di gerai-gerai mitra yang akan dipertanyakan integritasnya namun data transaksi di semua pihak akan diragukan integritasnya, maka dari itu diperlukan suatu sistem yang dapat menjaga integritas data baik di sisi *server* gerai, *server* kereta maupun pada saat pengiriman antar keduanya.

Berdasarkan fungsi yang telah dijelaskan sebelumnya, sistem membutuhkan 3 sub-sistem yang pertama sistem untuk menangani transaksi dan mengelola *database server* gerai, sistem untuk menangani data kiriman dari *server* gerai dan mengelola *server* kereta serta sistem untuk mengubah pesan menjadi *digest* untuk menunjang kinerja dua sistem yang lain seperti yang terlihat pada Gambar 4.1. Ketiga sub-sistem akan saling terhubung dengan skema sub-sistem *server* gerai akan melakukan transaksi kemudian data transaksi akan di dirubah menjadi *digest* dengan sub-sistem *hash* untuk ditambahkan pada data transaksi guna dicatatkan pada *database* serta dikirimkan ke *server* kereta sebelum pengiriman data transaksi yang dimaksudkan untuk membandingkan apakah *digest* yang telah dikirim nilainya sama dengan hasil perhitungan *digest* data transaksi yang dikirimkan setelahnya agar integritas data transaksi yang dikirimkan tetap terjaga, dan sistem *server* kereta akan mencatatkan data yang dikirim oleh *server* gerai pada *database* dengan ditambahkan *digest* dari data transaksi. Seluruh sub-sistem dibangun dengan bahasa pemrograman Python agar memudahkan pemrograman jaringan guna menunjang kinerja sistem.



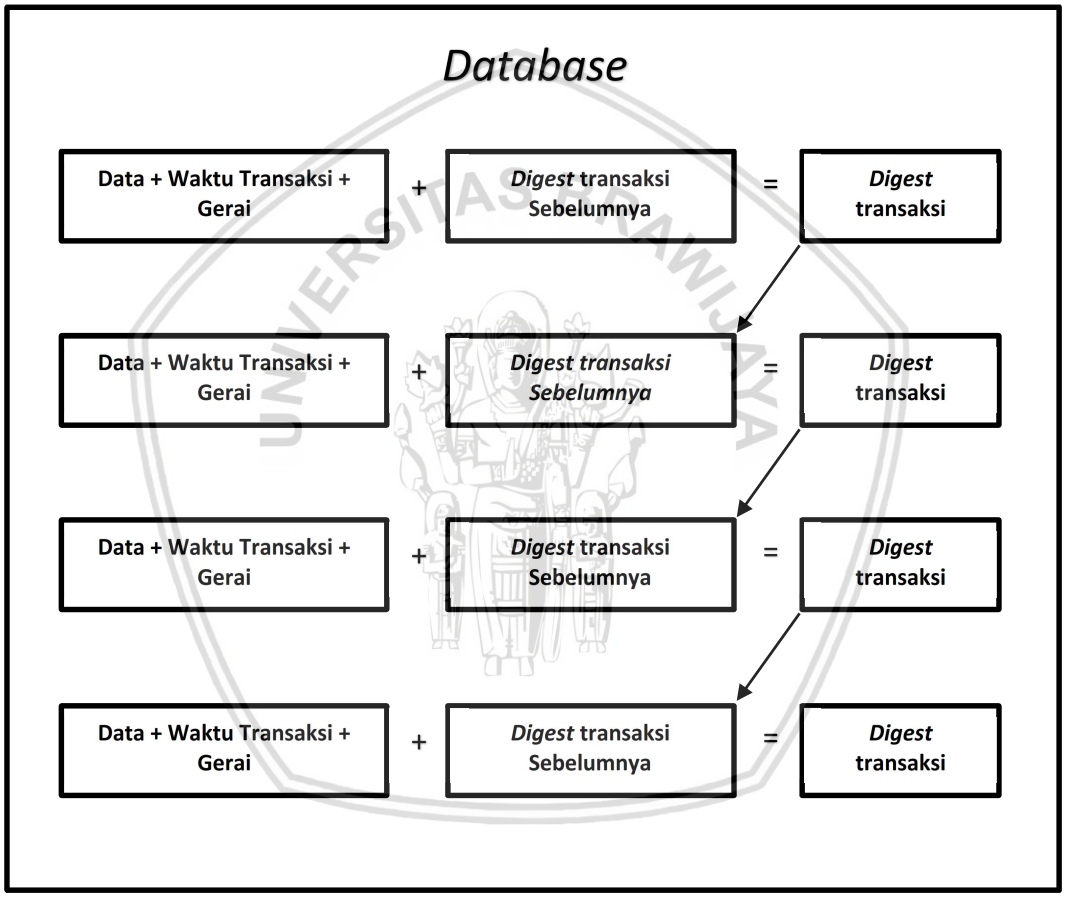

**Gambar 4.2 Gambaran Sistem**

Saat ini jumlah gerai mitra kereta api tidaklah sedikit sehingga nantinya sistem untuk gerai dibuat tidak terperinci dan bersifat umum agar dapat merepresentasikan banyak gerai penjual tiket kereta. Nantinya saat gerai-gerai melakukan transaksi, data akan disimpan kedalam *database* masing-masing dan dikirimkan ke *server* kereta api untuk disimpan dalam *database server* kereta api, itu artinya *database* kereta api akan mencatat data dari semua penjualan berbagai gerai tidak seperti *database* gerai yang hanya mencatat transaksi yang dilakukan gerai itu sendiri. Untuk menangani koneksi sistem gerai yang jumlahnya lebih dari satu maka sistem kereta akan bertindak sebagai *server* dan sistem gerai akan bertindak sebagai *client*, *server* akan menggunakan *protocol tcp* dan dibekali kemampuan multi-proses agar dapat menangani permintaan koneksi dari beberapa *client* dalam waktu bersamaan.




**Gambar 4.3 *Database* Sistem**

Cara kerjanya ialah data transaksi yang dihasilkan oleh proses transaksi sebelumnya akan ditambahkan cacatan waktu kapan transaksi itu berlangsung dan gerai mana yang melakukan transaksi serta nilai *digest* dari transaksi yang terakhir kali dilakukan, kemudian data tersebut akan dihitung nilai *digest*. Jika nilai *digest* telah ditemukan maka semua data akan dicatatkan pada baris baru *database* seperti yang terlihat pada gambar Gambar 4.2. catatan baru akan mengandung nilai *digest* catatan sebelumnya, jadi jika sedikit saja nilai data transaksi berubah maka nilai *digest* semua data-data transaksi setelahnya juga akan ikut berubah, hal ini bertujuan agar integritas data dapat terjaga dan jika ada manipulasi data dapat diketahui dengan mudah.

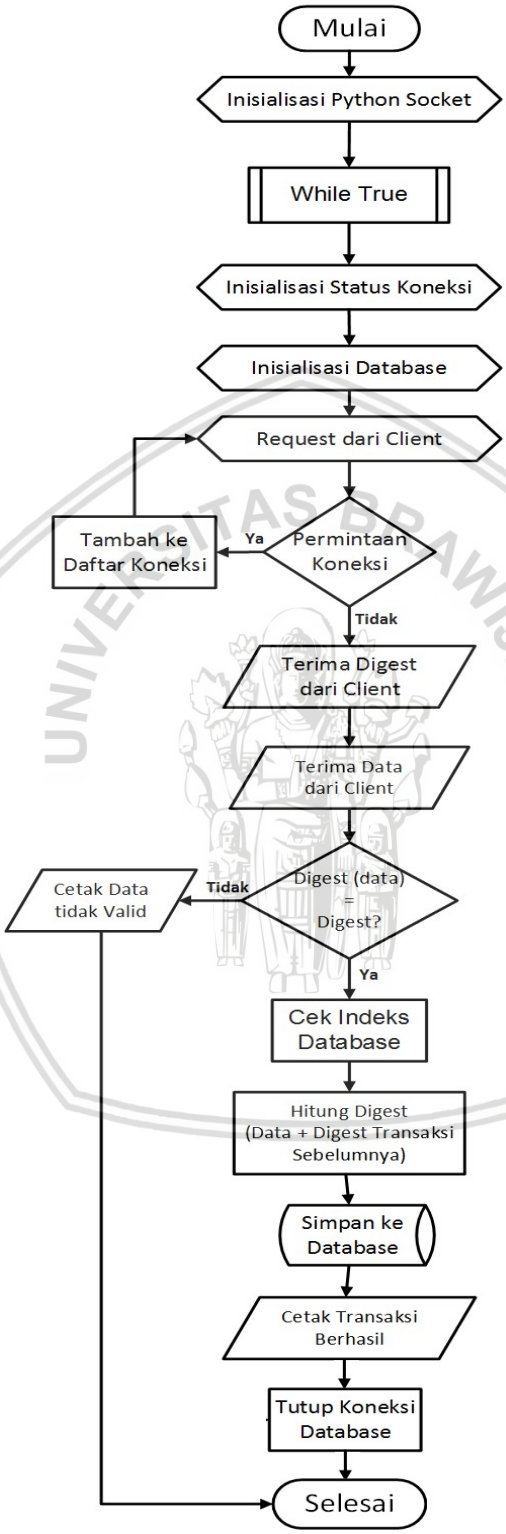



**Gambar 4.4 Cara Kerja Sistem**

### 4.2.2 Sistem *Server* Kereta Api

Sistem dibangun dengan bahasa pemrograman Python dan beberapa *library* diperlukan di antaranya *socket* untuk pemrograman jaringan, *select* untuk multi proses dan PyMySQL untuk mengelola *database* MySQL. Sistem akan melakukan perulangan untuk menunggu jika ada *client* yang akan melakukan permintaan koneksi, jika permintaan koneksi bersifat *three way hand shaking* maka *server* akan memasukan *client* tersebut ke daftar, jika permintaan koneksi bersifat

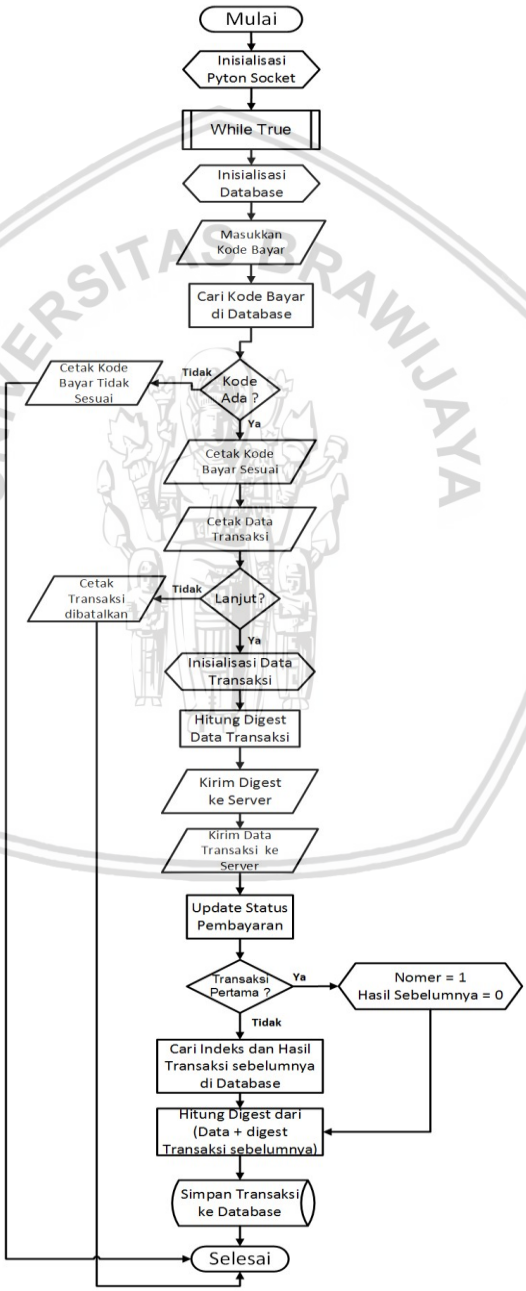
permintaan layanan makan *server* akan menampung pengiriman data dan mengolah data, jika data berhasil diolah maka *server* akan mengirimkan pesan berhasil dan jika data gagal diolah maka *server* akan mengirimkan pesan gagal.




**Gambar 4.5 Diagram alir Sistem *Server* Kereta**

### 4.2.3 Sistem *Server* Gerai

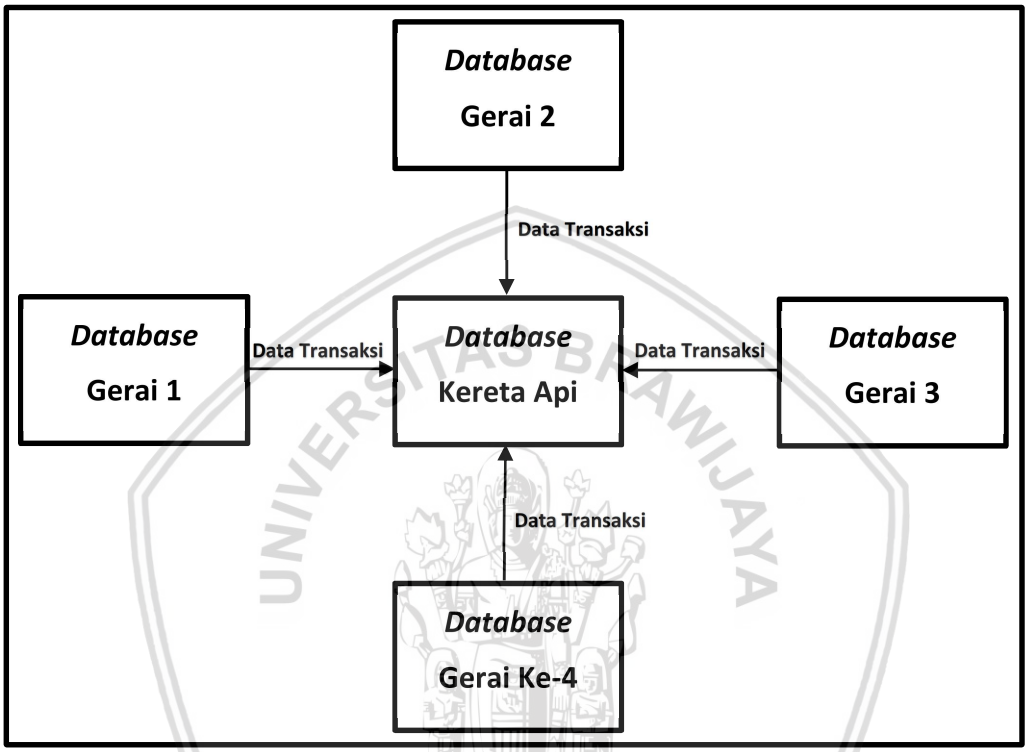
Sistem hampir sama dengan *Server* Kereta namun yang membedakannya ialah alur kerjanya dan tanpa *library select*, sistem akan melakukan perulangan untuk menunggu *client* yang akan melakukan transaksi, Sebelum melakukan transaksi dilakukan pengecekan terhadap kode bayar, jika kode bayar sesuai maka data transaksi akan dikirimkan ke *server* kereta dengan terlebih dahulu ditambahkan perhitungan *hash*, waktu transaksi dan nama gerai, jika transaksi dinyatakan berhasil oleh *server* kereta maka data transaksi beserta perhitungan *digest* akan dicatat dan sistem akan mencetak pesan transaksi berhasil.




**Gambar 4.6 Diagram alir Sistem *Server* Gerai**

## 4.3 Perancangan Pengujian

Setelah program dilakukan implementasi, penulis akan melakukan pengujian untuk memastikan sistem yang telah dibangun dapat berjalan dengan baik dan benar, ada empat parameter yang nantinya akan penulis ujikan terhadap sistem, yang pertama adalah ketepatan hasil perhitungan *digest* dari modul *SHA-1* dengan cara membandingkan hasil perhitungan program dengan *test vector* resmi.




**Gambar 4.7 Perancangan Pengujian**

Pengujian yang kedua ialah pengujian terhadap fungsionalitas sistem, karena sistem ini adalah sistem pengamanan data transaksi pada *database* terdistribusi maka penguji akan mencoba melakukan transaksi dari empat gerai seperti yang terlihat pada Gambar 4.6 yang nantinya keempat gerai tersebut akan mendistribusikan data transaksinya ke *server* kereta, penulis akan menilai apakah sistem dapat mengamankan data transaksi sebagaimana telah dijelaskan sebelumnya atau sistem gagal menjalankan fungsinya tersebut. pengujian ketiga adalah pengujian *quality of services*, pengujian ini akan mengukur kualitas dari layanan sistem, kemudian pengujian keempat dan yang terakhir adalah pengujian validitas data transaksi, karena tujuan dari sistem ini ialah menjaga integritas data transaksi maka validitas data transaksi perlu diuji untuk membuktikan bahwa sistem ini dapat mencapai tujuannya dan catatan data transaksi yang dihasilkan terbukti valid.

# BAB 5 IMPLEMENTASI

Setelah sebelumnya penulis membuat rancangan sebagai acuan untuk implementasi, pada bab ini penulis akan melakukan implementasi rancangan tersebut kedalam bahasa pemrograman Python. Sistem terdiri dari tiga sub-sistem yaitu sistem *Secure Hash Algorithm* 1 (*SHA-1*), sistem *Server* Kereta, Sistem *Server* Gerai yang ketiganya jika disatukan akan membentuk Sistem Pengamanan Data Transaksi Pada *Database* Terdistribusi.

## 5.1 *Source Code Secure Hash Algorithm* 1 (*SHA-1*)

Kode program ini adalah implementasi dari algoritme *SHA-1* yang dipublikasikan oleh National Institute *of Standards* and Technology (*NIST*) yang ditulis dengan bahasa pemrograman Python. Berfungsi sebagai modul *hashing*

Pertama-tama deklarasikan variabel H0 sampai H4 dengan nilai yang telah ditentukan oleh pembuat algoritme, untuk merubah sebuah pesan menjadi *digest* dengan panjang yang selalu tetap. Pertama-tama pesan akan diambil dari parameter *message* pada fungsi *GenerateDigest* kemudian akan dirubah kedalam bentuk biner kemudian akan ditambahkan angka satu dibelangkan kemudian pesan akan digenapkan menjadi kelipatan 512 *mod* 448 dengan menambahkan angka nol serta ditambahkan biner asli dengan format 64 bit sehingga pesan menjadi kelipatan 512, proses ini disebut dengan *padding*. Setelah pesan dilakukan *padding* kemudian pesan akan dibagi menjadi 80 bagian dengan bagian pertama sampai enam belas merupakan urutan pertama pesan dengan panjang 32 bit sedangkan urutan tujuh belas sampai delapan puluh adalah hasil perhitungan fungsi ROTL yang telah dideklarasikan sebelumnya dan telah ditentukan oleh pembuat algoritme. Setelah pesan dibagi menjadi 80 bagian. Kemudian dideklarasikan variabel a, b, c, dan d dengan nilai dari variabel H1, H2, H3 dan H4 untuk kemudian dihitung dengan rumus pada baris ke tujuh puluh tujuh sampai delapan puluh dua dengan menyertakan fungsi *fuction* dan fungsi *constant* yang telah di deklarasikan sebelumnya dan telah ditentukan oleh pembuat algoritme. Kemudian hasil dari perhitungan akan dimasukan kembali kedalam variabel H0 sampai H4 seperti pada baris delapan puluh empat sampai delapan puluh delapan. kemudian nilai dari variabel H0 sampai H4 akan digabungkan sehingga diperoleh niai *digest*.

Algoritme 1 : Fungsi *SHA-1*

```
1   def GenerateDigest(message):
2
3       H0 = 0x67452301
4       H1 = 0xEFCDAB89
5       H2 = 0x98BADCFE
6       H3 = 0x10325476
7       H4 = 0xC3D2E1F0
8
9       #Ubah Message kedalam bentuk biner
10      biner = ""
11
```

```
    biner = biner + (''.join("{0:08b}".format(ord(x), 'b') for x
in message))

    #Tambah 1 dibelakang pesan yang telha diubah kedalam biner
    padded = biner+"1"

    #Tambah angka 0
    While True
        padded=padded+"0"
            if (len(padded)%512 == 448%512):
                break

    #Tambah 64bit pesan asli
    padded=padded+'{0:064b}'.format(len(biner))

        #Function
    def funct(x, y, z, i):
        if 0 <= i <= 19:
            f = (x & y) ^ ((~x) & z)
        elif 20 <= i <= 39:
            f = x ^ y ^ z
        elif 40 <= i <= 59:
            f = (x & y) ^ (x & z) ^ (y & z)
        elif 60 <= i <= 79:
            f = x ^ y ^ z
        return f

        #Constants
    def K(i):
        if 0 <= i <= 19:
            k = 0x5A827999
        elif 20 <= i <= 39:
            k = 0x6ED9EBA1
        elif 40 <= i <= 59:
            k = 0x8F1BBCDC
        elif 60 <= i <= 79:
            k = 0xCA62C1D6
        return k

        #Sn(X)
    def ROTL(X, n):
        rotate = (( X << n) | (X >> (32 - n))) & 0xffffffff
        return rotate

    def split(l, n):
        s = [l[i:i+n] for i in xrange(0, len(l), n)]
        return s

    for i in split(padded, 512):
        w = [0]*80
        M = split(i, 32)
        for t in range(0, 80):
            if t <= 15 :
                #Mt(i)
                w[t] = int(M[t], 2)
            else :
                #ROTL1=( tt3^tt8^tt14^tt16 )
```

```
                w[t] = ROTL((w[t-3] ^ w[t-8] ^ w[t-14] ^ w[t-16]),
1)

        #Initialize the five working variabels
        a = H0
        b = H1
        c = H2
        d = H3
        e = H4

        for i in range(0, 80):
            T = ROTL(a,5) + funct(b,c,d,i) + e + K(i) + w[i] &
0xffffffff
            e = d
            d = c
            c = ROTL(b,30)
            b = a
            a = T

        H0 = a + H0 & 0xffffffff
        H1 = b + H1 & 0xffffffff
        H2 = c + H2 & 0xffffffff
        H3 = d + H3 & 0xffffffff
        H4 = e + H4 & 0xffffffff


        Digest                                                  =
'%08x'%H0+'%08x'%H1+'%08x'%H2+'%08x'%H3+'%08x'%H4

    return Digest
```

## 5.2 *Source Code Server* Kereta

Program *Server* Kereta berfungsi untuk mengelola data transaksi serta mengelola *database*. Program ini nantinya dijalankan pada *server* Kereta Api, dalam penelitian ini penulis menggunakan MySQL sebagai *database* dan PyMySQL sebagai *library* pengelola *database* karena program ini menggunakan bahasa pemrograman Python serta *library select* untuk multi proses agar *server* dapat melayani beberapa *client* secara bersamaan, komunikasi antara *client* dan *server* menggunakan *protocol tcp* dan diasumsikan alamat IP dari *server* adalah 192.168.56.101 dan komunikasi menggunakan port 7500. Pada implementasi sesungguhnya bisa jadi *database* yang digunakan atau alamat *server* yang digunakan berbeda, dalam penelitian ini penulis hanya melakukan simulasi untuk keperluan ilmu pengetahuan.

Cara kerja program ini ialah akan memanfaatkan *library socket*, *select* Python sebagai penunjang komunikasi jaringan kemudian *library* PyMySQL untuk mengelola *database* MySQL dan juga memanfaatkan program *SHA-1* sebagai penghitung *digest* dari data transaksi, program ini akan menunggu permintaan koneksi dari *server* gerai yang akan melakukan transaksi, ketika ada *server* gerai yang meminta koneksi maka *server* kereta akan mengizinkan dan menambahkan *server* gerai tersebut kedalam daftar koneksi, *server* kereta akan melakukan koneksi dengan *database* untuk mempersiapkan transaksi dan koneksi dari *server*

gerai akan segera dilayani. *Server* kereta akan menunggu pengiriman data transaksi dari *server* gerai, pertama-tama *server* gerai akan mengirim hasil perhitungan *digest* dari data transaksi terlebih dahulu, setelah *digest* disimpan oleh *server* kereta kemudian data transaksi akan dikirim dan nantinya data transaksi akan dihitung ulang oleh *server* kereta serta dicocokan hasil perhitungannya dengan *digest* yang telah disimpan sebelumnya, hal ini mencegah adanya manipulasi data saat data transaksi di transmisikan. Kemudian *server* kereta akan memeriksa indeks dari *database* untuk menentukan nomor catatan baru dan menentukan *hashing* catatan sebelumnya untuk menentukan *hashing* dari data transaksi yang sedang diproses, *server* kereta akan memanggil program *SHA-1* untuk menghitung *digest* dari data transaksi beserta *digest* data transaksi sebelumnya dan mencatatkannya kedalam *database*. Setelah transaksi selesai dicatatkan maka *server* kereta akan memberikan konfirmasi kepada *server* gerai bahwa transaksi telah berhasil dilakukan.

Algoritme 2 : Fungsi *Server* Kereta

```
1   import socket
2   import select
3   import pymysql
4   from sha1 import GenerateDigest
5
6   sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7
8   sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
9
10  sock.bind( ('', 7500) )
11
12  sock.listen(10)
13
14  list_koneksi = [sock]
15
16  def send_to_all_clients(conn, msg):
17      for client in conn.clients :
18          client.connection.send(msg)
19
20  while True :
21      inputready,          outputready,          errorready          =
    select.select(list_koneksi, [], [])
22
23
24      for n in inputready :
25          db                                                          =
    pymysql.connect("localhost","root","br89q","transaksi" )
26          cursor = db.cursor()
27
28          if n == sock :
29              conn, addr = sock.accept()
30              list_koneksi.append(conn)
31          else :
32              try :
33                  data = n.recv(100)
34                  try :
35                      hash1 = data
36                      n.send("Sedang Diproses oleh Server.....")
```

```
37                      print "Transaksi Sedang Dilakukan"
38                      data = n.recv(250)
39
    nama,id,tipe,tempatduduk,nomortiket,kereta,kelas,
    berangkat,tiba,statusbayar,harga,kodebayar,waktubayar,gerai    =
    data.split("$?!")
40                      print (gerai+" Memulai Transaksi")
41                      hash2                                          =
    GenerateDigest(nama+id+tipe+tempatduduk+
    nomortiket+kereta+kelas+berangkat+tiba+statusbayar+harga+kodebay
    ar+waktubayar+gerai)
42                      if hash1==hash2:
43                          hashsebelum = ""
44
45                          sql    =    "SELECT    max(nomor)    FROM
    pembayaran;"
46                          cursor.execute(sql)
47                          b=cursor.fetchall()
48                          for row in b :
49                              nomor = (row[0])
50
51                          if nomor != None:
52                              sql = "SELECT * FROM pembayaran WHERE
    nomor=%s;"
53                              cursor.execute(sql,(nomor))
54                              b=cursor.fetchall()
55                              for row in b :
56                                  hashsebelum = (row[16])
57                              nomor= str(int(nomor)+1)
58
59                          elif nomor == None:
60                              nomor="1"
61                              hashsebelum = "0"
62
63                          hashsekarang=
    GenerateDigest(nomor+nama+id+tipe+
    tempatduduk+nomortiket+kereta+kelas+berangkat+tiba+statusbayar+h
    arga+kodebayar+waktubayar+gerai+hashsebelum)
64                          sql = "INSERT INTO pembayaran (nomor,
    nama, id, tipe, tempatduduk, nomortiket, kereta, kelas, berangkat,
    tiba,   statusbayar,   harga,   kodebayar,   waktubayar,   gerai,
    hashsebelum,                    hash)                    VALUES
    (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"
65
    cursor.execute(sql,((nomor,nama,id,tipe,tempatduduk,
    nomortiket,kereta,kelas,berangkat,tiba,statusbayar,harga,kodebay
    ar,waktubayar,gerai,hashsebelum,hashsekarang)))
66                          db.commit()
67                          n.send("Transaksi Berhasil")
68                          print ("Transaksi "+gerai+" Berhasil")
69
70                      else :
71                          print "Integritas Data Tidak Dipercaya
    Transaksi Dibatalkan"
72
73                  except ValueError:
74                      list_koneksi.remove(n)
```

```
75                      n.close()
76                      print ("Terjadi Kesalahan, Koneksi ke client
"+gerai+" diputus")
77              except socket.error:
78                  list_koneksi.remove(n)
79                  n.close()
80                  print ("Koneksi ke client "+gerai+" diputus")
81      db.close()
```

## 5.3 *Source Code Server* Gerai

Program *Server* Gerai berfungsi untuk melakukan transaksi dari data pemesanan serta mengelola *database*. Program ini nantinya dijalankan pada *server* gerai-gerai mitra penjualan, dalam penelitian ini penulis menggunakan MySQL sebagai *database* dan PyMySQL sebagai *library* pengelola *database* karena program ini menggunakan bahasa pemrograman Python, komunikasi antara *client* dan *server* menggunakan *protocol tcp* dan diasumsikan alamat IP dari *server* adalah 192.168.56.102 sampai 192.168.56.254 sesuai kebutuhan dan komunikasi menggunakan port 7500. Pada implementasi sesungguhnya bisa jadi *database* yang digunakan atau alamat *server* yang digunakan berbeda, dalam penelitian ini penulis hanya melakukan simulasi untuk keperluan ilmu pengetahuan.

Cara kerja program ini ialah akan memanfaatkan *library socket* Python sebagai penunjang komunikasi jaringan kemudian *library* MySQL untuk mengelola *database* MySQL dan juga memanfaatkan program *SHA-1* sebagai penghitung *digest* dari data transaksi, inisialisasi koneksi diatur agar program dapat berkomunikasi dengan *server* kereta dengan menuliskan alamat ip dari *server* kereta, saat program dijalankan maka koneksi ke *server* kereta akan dibuat kemudian program akan melakukan koneksi ke *database* setelah itu program akan meminta pengguna memasukan kode bayar ketika akan melakukan transaksi, program akan mencari data pemesanan yang memiliki kode program sama dengan kode program yang dimasukan pengguna, jika kode program ada maka program akan mencetak data pemesanan untuk klarifikasi namun jika kode bayar tidak ditemukan maka program akan membatalkan transaksi. Setelah pengguna melakukan konfirmasi bahwa data transaksi benar dan bersedia melanjutkan transaksi maka *server* gerai akan menghitung *digest* dari data transaksi untuk dikirimkan ke *server* kereta kemudian data transaksi akan dikirimkan pada transmisi berikutnya. Program akan menerima status pembayaran dari *server* kereta dan *server* gerai akan mencetak status tersebut. Jika pembayaran dinyatakan berhasil oleh *server* kereta dan data telah dicatat pada *database* maka data transaksi akan dihitung sebelum dicatatkan pada *database server* gerai dengan nomor transaksi baru dan tambahan *hashing* dari catatan sebelumnya serta hasil *hashing* itu sendiri. Setelah data transaksi berhasil dicatatkan pada *database server* gerai maka program akan mencetak pesan transaksi berhasil.

Algoritme 3 : Fungsi *Server* Gerai

```
1   import socket
2   import pymysql
3   from sha1 import GenerateDigest
4   from datetime import datetime
5
6   sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7
8   sock.connect( ('192.168.56.101', 7500) )
9
10  while True :
11      db = pymysql.connect("localhost","root","br89q","transaksi" )
12      cursor = db.cursor()
13
14      kode = raw_input("Masukan Kode Bayar : ")
15      sql = "SELECT * FROM pemesanan WHERE kodebayar ='%s';" %
    kode.strip()
16      cursor.execute(sql)
17      a=cursor.fetchall()
18      kodebayar=""
19      gerai="Gerai-01"
20      for row in a :
21          nomor,nama,id,tipe,tempatduduk,nomortiket,kereta,kelas,b
    erangkat,tiba,statusbayar,harga,kodebayar=(row[0]),(row[1]),(row
    [2]),(row[3]),(row[4]),(row[5]),(row[6]),(row[7]),(row[8]),(row[
    9]),(row[10]),(row[11]),(row[12])
22      if kode == kodebayar :
23          print ""
24          print "Kode Bayar Sesuai"
25          print ""
26          print ("Pembayaran Atas Nama : "+ nama)
27          print ("Jumlah Pembayaran : "+ harga)
28          print ""
29          konfirmasi = raw_input("Lanjutkan Transaksi ? (y/t) : ")
30          print ""
31          if konfirmasi == "y":
32              waktu = str(datetime.now())
33              data                                                =
34  (nama+id+tipe+tempatduduk+nomortiket+kereta+kelas+berangkat+tiba
    +statusbayar+harga+kodebayar+waktu+gerai)
35              digest =GenerateDigest(data)
36              sock.send(digest)
37              data = sock.recv(100)
38              print data
39              print""
40              data  =  nama+  "$?!"  +id+  "$?!"  +tipe+  "$?!"
    +tempatduduk+ "$?!" +nomortiket+ "$?!" +kereta+ "$?!" +kelas+
    "$?!" +berangkat+ "$?!" +tiba+ "$?!" +statusbayar+ "$?!" +harga+
    "$?!" +kodebayar+ "$?!" +waktu+ "$?!" +gerai
41              sock.send(data)
42              data = sock.recv(100)
43              print data
44              sql  =  "UPDATE  pemesanan  SET  statusbayar='Sudah
    Dibayar' WHERE nomor= (%s);" % nomor.strip()
45              cursor.execute(sql)
```

```
46              db.commit()
47
48              sql = "SELECT max(nomor) FROM pembayaran;"
49              cursor.execute(sql)
50              b=cursor.fetchall()
51              for row in b :
52                  nomor = (row[0])
53
54              if nomor != None:
55                  sql = "SELECT * FROM pembayaran WHERE nomor=%s;"
56                  cursor.execute(sql,(nomor))
57                  b=cursor.fetchall()
58                  for row in b :
59                      hashsebelum = (row[16])
60                  nomor= str(int(nomor)+1)
61
62              elif nomor == None:
63                  nomor="1"
64                  hashsebelum = "0"
65
66              hashsekarang=    GenerateDigest(nomor+nama+id+tipe+
    tempatduduk+nomortiket+kereta+kelas+berangkat+tiba+statusbayar+h
    arga+kodebayar+waktu+gerai+hashsebelum)
67              sql = "INSERT INTO pembayaran (nomor, nama, id, tipe,
    tempatduduk,  nomortiket,  kereta,  kelas,  berangkat,  tiba,
    statusbayar, harga, kodebayar, waktubayar, gerai, hashsebelum,
    hash)                                                    VALUES
    (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"
68              cursor.execute(sql,((nomor,nama,id,tipe,tempatduduk,
    nomortiket,kereta,kelas,berangkat,tiba,”Sudah
    Dibayar”,harga,kodebayar,waktu,gerai,hashsebelum,hashsekarang)))
69              db.commit()
70
71          elif konfirmasi == "t":
72              print "Transaksi Dibatalkan"
73          else :
74              print "Pilihan Salah, Transaksi Dibatalkan"
75      else :
76          print "Kode Bayar Salah"
77
78      print""
79      print""
80      db.close()
```

# BAB 6 PENGUJIAN DAN PEMBAHASAN

Pada bab ini penulis akan melakukan pengujian terhadap sistem yang telah dirancang dan dilakukan implementasi sebelumnya, penulis akan melakukan tiga pengujian, pengujian pertama ialah pengujian *test vector*, yang kedua ialah pengujian fungsionalitas, dan yang ketiga ialah pengujian *Quality of Sevices*.

## 6.1 Pengujian *Test Vector*

Pengujian *Test Vector* bertujuan untuk memastikan sub-program *SHA-1* berjalan dengan baik dan benar serta menghasilkan keluaran sesuai dengan hasil perhitungan pembuat algoritme, sebagai pembanding, penulis menggunakan *Test Vector* resmi dari National Institute *of Standards* and Technology (*NIST*) yang telah penulis lampirkan pada halaman lampiran.

Dalam pengujian ini penulis akan menjalankan program dengan masukan yang sama seperti *test vector*, kemudian akan membandingkan hasil perhitungan program dengan *test vector* namun sebelum program dijalankan penulis melakukan modifikasi pada kode program *SHA-1* agar program dapat mencetak hasil perhitungan tiap langkah, berikut adalah kode program yang telah penulis modifikasi:

Algoritme 4 : Fungsi *SHA-1* Modifikasi

```
1   def GenerateDigest(message):
2
3       H0 = 0x67452301
4       H1 = 0xEFCDAB89
5       H2 = 0x98BADCFE
6       H3 = 0x10325476
7       H4 = 0xC3D2E1F0
8
9       #Ubah Message kedalam bentuk biner
10      biner = ""
11      biner = biner + (''.join("{0:08b}".format(ord(x), 'b') for x
    in message))
12
13      #Tambah 1 dibelakang pesan yang telha diubah kedalam biner
14      padded = biner+"1"
15
16      #Tambah angka 0
17      While True
18          padded=padded+"0"
19              if (len(padded)%512 == 448%512):
20                  break
21
22      #Tambah 64bit pesan asli
23      padded=padded+'{0:064b}'.format(len(biner))
24
25          #Function
```

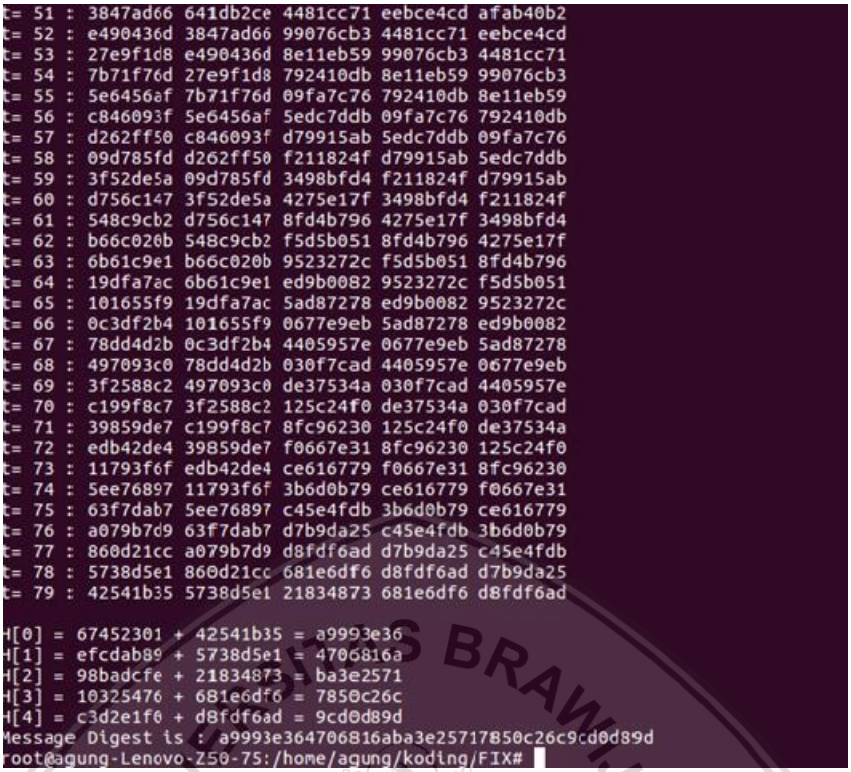
```
26      def funct(x, y, z, i):
27          if 0 <= i <= 19:
28              f = (x & y) ^ ((~x) & z)
29          elif 20 <= i <= 39:
30              f = x ^ y ^ z
31          elif 40 <= i <= 59:
32              f = (x & y) ^ (x & z) ^ (y & z)
33          elif 60 <= i <= 79:
34              f = x ^ y ^ z
35          return f
36
37          #Constants
38      def K(i):
39          if 0 <= i <= 19:
40              k = 0x5A827999
41          elif 20 <= i <= 39:
42              k = 0x6ED9EBA1
43          elif 40 <= i <= 59:
44              k = 0x8F1BBCDC
45          elif 60 <= i <= 79:
46              k = 0xCA62C1D6
47          return k
48
49          #Sn(X)
50      def ROTL(X, n):
51          rotate = (( X << n) | (X >> (32 - n))) & 0xffffffff
52          return rotate
53
54      def split(l, n):
55          s = [l[i:i+n] for i in xrange(0, len(l), n)]
56          return s
57
58      for i in split(padded, 512):
59          w = [0]*80
60          M = split(i, 32)
61          print(‘’)
62          print(“Blok Contents :”)
63          for t in range(0, 80):
64              if t <= 15 :
65                  #Mt(i)
66                  w[t] = int(M[t], 2)
67                  print("W["+str(t)+"] = "+('%08x'%w[t]))
68
69              else :
70                  #ROTL1=( tt3^tt8^tt14^tt16 )
71                  w[t] = ROTL((w[t-3] ^ w[t-8] ^ w[t-14] ^ w[t-16]),
    1)
72
73          #Initialize the five working variabels
74          a = H0
75          b = H1
76          c = H2
77          d = H3
78          e = H4
79
80          print(‘’)
81
```

```
82          for i in range(0, 80):
83              T = ROTL(a,5) + funct(b,c,d,i) + e + K(i) + w[i] &
    0xffffffff
84              e = d
85              d = c
86              c = ROTL(b,30)
87              b = a
88              a = T
89              print("t= "+str(i)+" : "+('%08x'%a)+" "+('%08x'%b)+"
    "+('%08x'%c)+" "+('%08x'%d)+" "+('%08x'%e))
90
91         h0,h1,h2,h3,h4=H0,H1,H2,H3,H4
92
93          H0 = a + H0 & 0xffffffff
94          H1 = b + H1 & 0xffffffff
95          H2 = c + H2 & 0xffffffff
96          H3 = d + H3 & 0xffffffff
97          H4 = e + H4 & 0xffffffff
98
99          Digest                                                  =
    '%08x'%H0+'%08x'%H1+'%08x'%H2+'%08x'%H3+'%08x'%H4
100
101         print('')
102         print("H[0]   =   "+('%08x'%h0)+"   +   "+('%08x'%a)+"   =
    "+('%08x'%H0))
103         print("H[1]   =   "+('%08x'%h1)+"   +   "+('%08x'%b)+"   =
    "+('%08x'%H1))
104         print("H[2]   =   "+('%08x'%h2)+"   +   "+('%08x'%c)+"   =
    "+('%08x'%H2))
105         print("H[3]   =   "+('%08x'%h3)+"   +   "+('%08x'%d)+"   =
    "+('%08x'%H3))
106         print("H[4]   =   "+('%08x'%h4)+"   +   "+('%08x'%e)+"   =
    "+('%08x'%H4))
107
108     return Digest
109
110 print("Message      Digest      is      :      "+(GenerateDigest
    ("abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq ")))
```

### 6.1.1 *Test Vector* Dengan Panjang Message Satu Blok

Dari dua *test vector* yang disediakan *NIST* penulis akan mencoba *test vector* dengan panjang *message* satu blok dahulu untuk dicocokan dengan hasil perhitungan program. *Test vector* memiliki masukan "abc" dan akan menghasilkan *digest* "A9993E364706816ABA3E25717850C26C9CD0D89D", nantinya program akan diberi masukan yang sama dengan *test vector* untuk dilihat apakah keluaran *digest* serta langkah penyelesaiannya sama atau tidak dengan *test vector*.


**Gambar 6.1 Hasil Perhitungan Satu Blok Bagian 1**




**Gambar 6.2 Hasil Perhitungan Satu Blok Bagian 2**

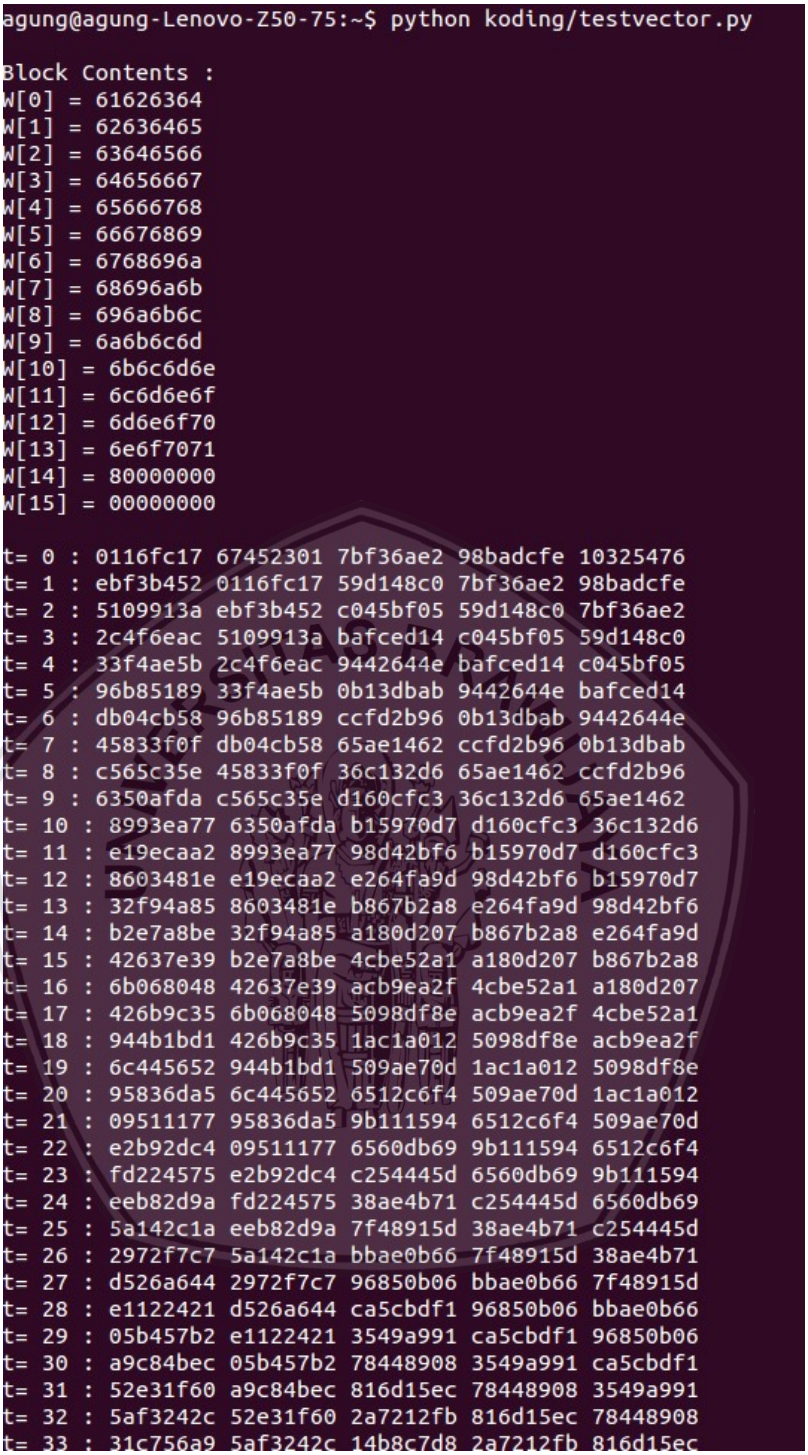
Setelah melakukan percobaan membandingkan hasil perhitungan program dengan *test vector* dapat dilihat hasil dari perhitungan sama dengan *test vector* dan dapat disimpulkan bahwa program telah sesuai dan dapat digunakan sebagai modul penghitung *SHA-1* pada sistem yang telah dibuat seperti yang dapat dilihat pada Tabel 6.1.

| objek | Masukan | Keluaran |
|---|---|---|
| *Test Vector* | abc | A9993E364706816ABA3E25717850C26C9CD0D89D |
| Hasil Keluaran | abc | A9993E364706816ABA3E25717850C26C9CD0D89D |

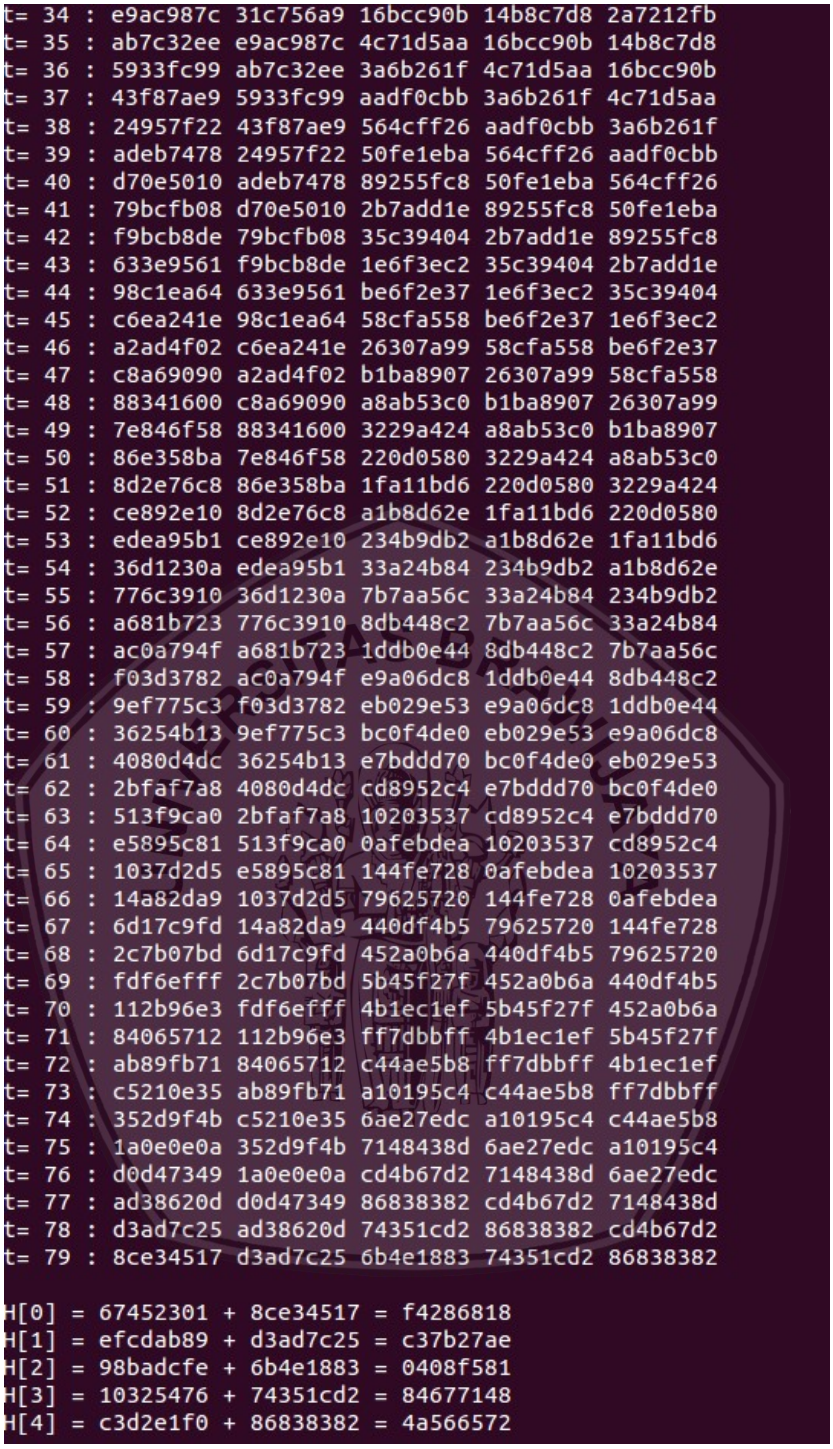
**Tabel 6.1 Hasil Pengujian Satu Blok**

### 6.1.2 *Test Vector* Dengan Panjang Message Dua Blok

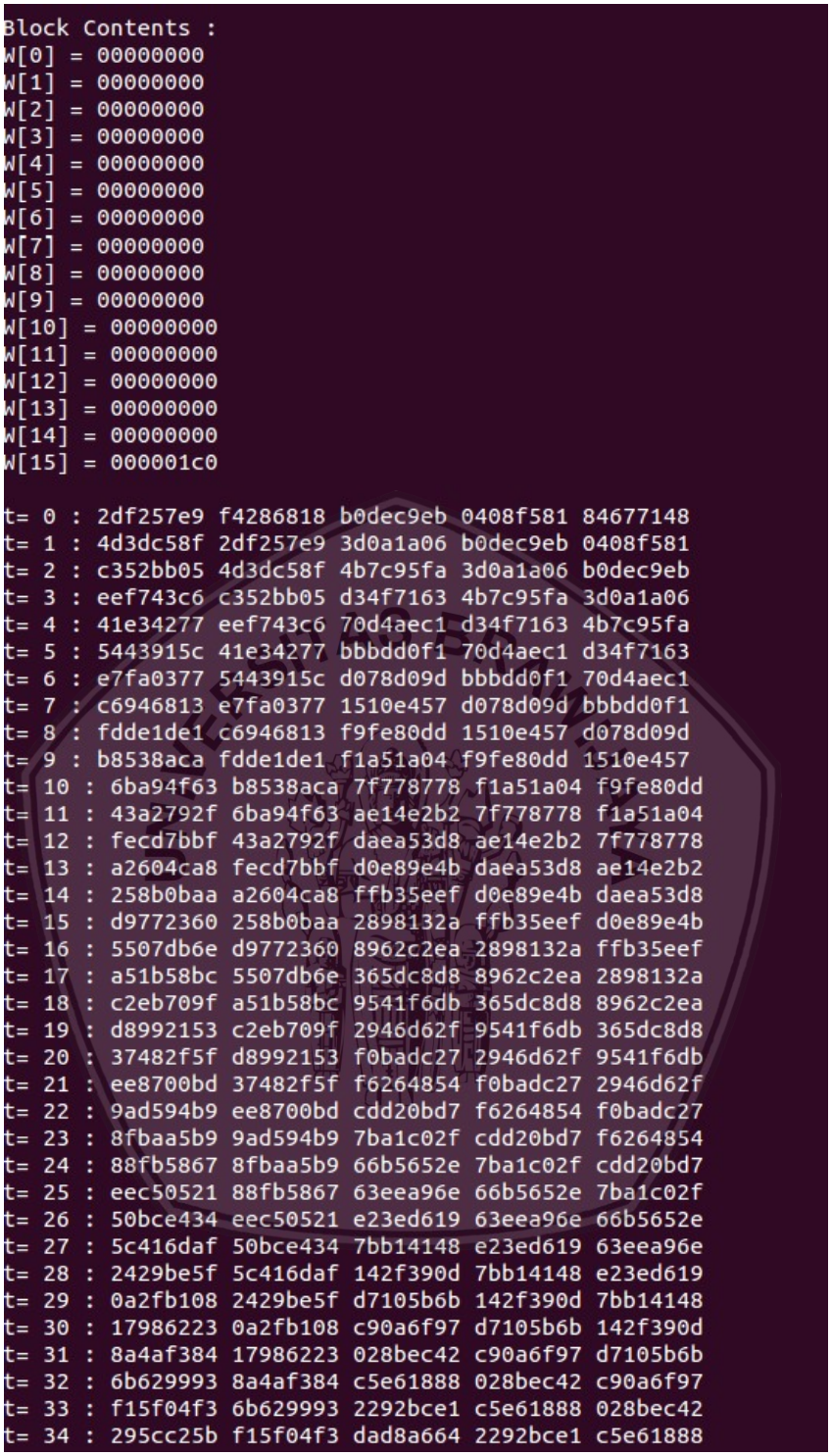
Setelah sebelumnya menguji dengan menggunakan *test vector* yang memiliki panjang *message* satu blok, kali ini penulis akan menguji dengan menggunakan *test vector* yang memiliki panjang *message* dua blok untuk cocokan dengan hasil perhitungan program. *Test vector* memiliki masukan "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" dan akan menghasilkan *digest* "84983E441C3BD26EBAAE4AA1F95129E5E54670F1", nantinya program akan diberi masukan yang sama dengan *test vector* untuk dilihat apakah keluaran *digest* serta langkah penyelesaiannya sama atau tidak dengan *test vector*.




**Gambar 6.3 Hasil Perhitungan Dua Blok Bagian 1**




**Gambar 6.4 Hasil Perhitungan Dua Blok Bagian 2**




**Gambar 6.5 Hasil Perhitungan Dua Blok Bagian 3**


**Gambar 6.6 Hasil Perhitungan Dua Blok Bagian 4**

Setelah melakukan percobaan membandingkan hasil perhitungan program dengan *test vector* dapat dilihat hasil dari perhitungan sama dengan *test vector* dan dapat disimpulkan bahwa program telah sesuai dan dapat digunakan sebagai modul penghitung *SHA-1* pada sistem yang telah dibuat dibuat seperti yang dapat dilihat pada Tabel 6.2.

| objek | Masukan | Keluaran |
|---|---|---|
| *Test Vector* | abcdbcdecdefdefgefghfghighijhij kijkljklmklmnlmnomnopnopq | 84983E441C3BD26EBAAE4AA 1F95129E5E54670F1 |
| Hasil Keluaran | Abcdbcdecdefdefgefghfghighijhij kijkljklmklmnlmnomnopnopq | 84983E441C3BD26EBAAE4AA 1F95129E5E54670F1 |

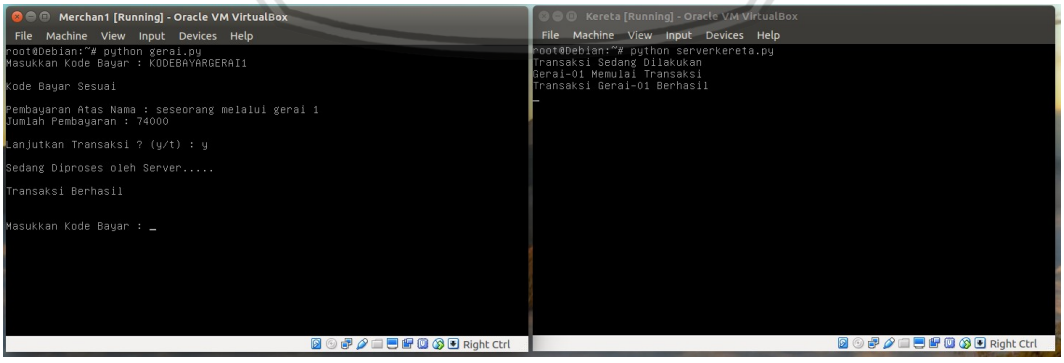
**Tabel 6.2 Hasil Pengujian Dua Blok**

## 6.2 Pengujian Fungsionalitas

Pengujian fungsionalitas adalah pengujian untuk mengetahui apakah sistem dapat menjalankan tugas sesuai fungsinya dengan baik dan benar atau sebaliknya, penulis melakukan pengujian fungsionalitas dengan cara melakukan transaksi dari empat *server* gerai yang berbeda kepada *server* kereta api kemudian dilihat apakah pembayaran dapat dicatatkan pada *server* kereta api dengan skema pengamanan atau sebaliknya. Percobaan akan dilakukan dengan lima mesin *virtual* dengan masing-masing empat mesin *virtual* untuk *server* gerai dan satu mesin *virtual* untuk *server* kereta, semua mesin menggunakan sistem operasi linux Debian 8. Berikut percobaan yang penulis lakukan:

### 6.2.1 Transaksi dari gerai 1

Transaksi akan dilakukan dari *server* gerai ke *server* kereta dengan asumsi tiket telah dipesan dan telah tercatat dalam *database* pemesanan pada *server* gerai, kemudian saat transaksi *server* gerai akan meminta kode bayar dan akan dicocokan dengan data pemesanan. Jika kode bayar ditemukan maka *server* gerai akan meminta konfirmasi untuk melanjutkan transaksi, saat transaksi dilanjutkan *server* gerai akan mengirimkan *digest* data transaksi dan data transaksi ke *server* kereta untuk disimpan dalam *database server* kereta dan *database* gerai sendiri dan diganti statusnya menjadi “Sudah Dibayar”, rangkaian transaksi tersebut telah penulis lakukan dan dapat dilihat pada Gambar 6.7 .




**Gambar 6.7 Transaksi Dari Gerai 1**

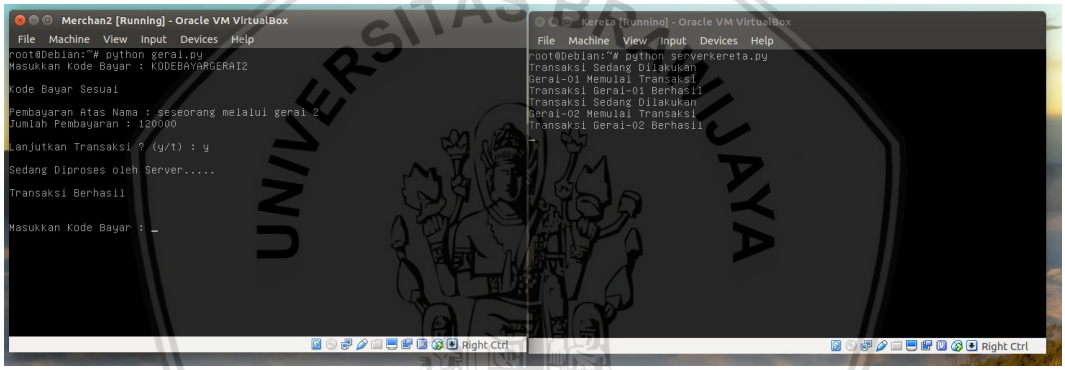
Sehingga akan menghasilkan cacatan transaksi dengan tambahan *digest* data transaksi pada *server* gerai sebagai berikut:

| nomer | nama | id | tipe | tempatduduk | nomertiket | kereta | kelas | berangkat | tiba | statusbayar | harga | kodebayar | waktubayar | gerai | hashsebelum | hash |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | seseorang melalui gerai 1 | 32454345345223 | DEWASA | 18 C | 4634532323254 | LOGAWA | EKONOMI | Bangil 09.00 | Lempuyangan 16.00 | Belum Dibayar | 74000 | KODEBAYARGERAI1 | 2018-05-18 03:22:42.860913 | Gerai-01 | 0 | a77b913dabd93bfd3ce900aaa207348c204d0874 |

**Gambar 6.8 *Database* Gerai 1**

### 6.2.2 Transaksi dari gerai 2

Transaksi akan dilakukan dari *server* gerai ke *server* kereta dengan asumsi tiket telah dipesan dan telah tercatat dalam *database* pemesanan pada *server* gerai, kemudian saat transaksi *server* gerai akan meminta kode bayar dan akan dicocokan dengan data pemesanan. Jika kode bayar ditemukan maka *server* gerai akan meminta konfirmasi untuk melanjutkan transaksi, saat transaksi dilanjutkan *server* gerai akan mengirimkan *digest* data transaksi dan data transaksi ke *server* kereta untuk disimpan dalam *database server* kereta dan *database* gerai sendiri dan diganti statusnya menjadi "Sudah Dibayar", rangkaian transaksi tersebut telah penulis lakukan dan dapat dilihat pada Gambar 6.9 .




**Gambar 6.9 Transaksi Dari Gerai 2**

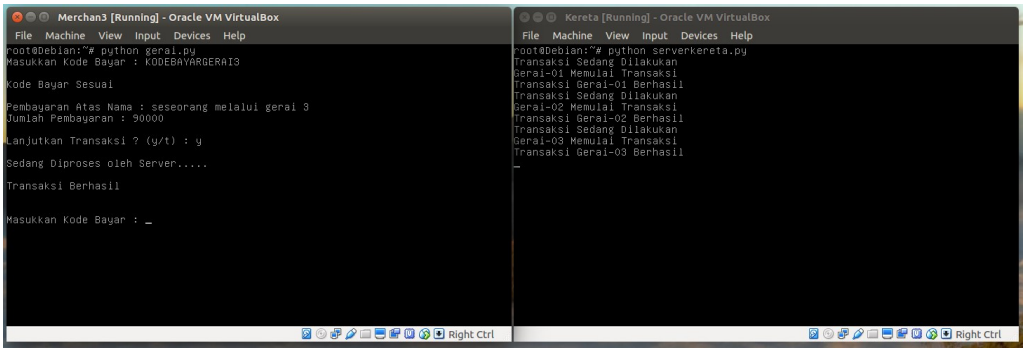
Sehingga akan menghasilkan cacatan transaksi dengan tambahan *digest* data transaksi pada *server* gerai sebagai berikut:

| nomer | nama | id | tipe | tempatduduk | nomertiket | kereta | kelas | berangkat | tiba | statusbayar | harga | kodebayar | waktubayar | gerai | hashsebelum | hash |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | seseorang melalui gerai 2 | 26546546345345223 | DEWASA | 11 A | 343554342346723765 | JAYABAYA | EKONOMI | Gubeng 21.00 | Tegal 04.00 | Belum Dibayar | 120000 | KODEBAYARGERAI2 | 2018-05-18 03:25:57.874419 | Gerai-02 | 0 | cc80f2a19668000b6f97dc3d34c4f4b065b1e6eb |

**Gambar 6.10 *Database* Gerai 2**

### 6.2.3 Transaksi dari gerai 3

Transaksi akan dilakukan dari *server* gerai ke *server* kereta dengan asumsi tiket telah dipesan dan telah tercatat dalam *database* pemesanan pada *server* gerai, kemudian saat transaksi *server* gerai akan meminta kode bayar dan akan dicocokan dengan data pemesanan. Jika kode bayar ditemukan maka *server* gerai akan meminta konfirmasi untuk melanjutkan transaksi, saat transaksi dilanjutkan *server* gerai akan mengirimkan *digest* data transaksi dan data transaksi ke *server* kereta untuk disimpan dalam *database server* kereta dan *database* gerai sendiri dan diganti statusnya menjadi "Sudah Dibayar", rangkaian transaksi tersebut telah penulis lakukan dan dapat dilihat pada Gambar 6.11 .




**Gambar 6.11 Transaksi Dari Gerai 3**

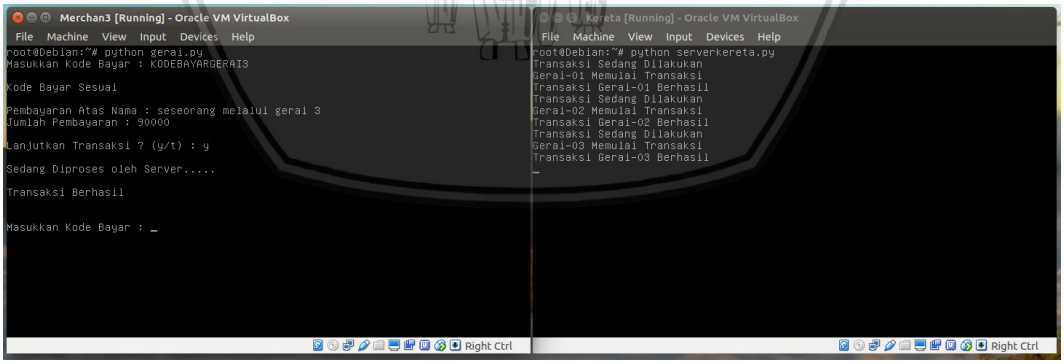
Sehingga akan menghasilkan cacatan transaksi dengan tambahan *digest* data transaksi pada *server* gerai sebagai berikut:

| nomer | nama | id | tipe | tempatduduk | nomertiket | kereta | kelas | berangkat | tiba | statusbayar | harga | kodebayar | waktubayar | gerai | hashsebelum | hash |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | seseorang melalui gerai 3 | 45546345345223 | DEWASA | 21 B | 564342346723765 | PASUNDAN | EKONOMI | Gubeng 06.00 | Kiaracondong 23.00 | Belum Dibayar | 90000 | KODEBAYARGERAI3 | 2018-05-18 03:27:13.508579 | Gerai-03 | 0 | 87db152ff26a8b8c6f51372d1f8df534adf7bb7f |

**Gambar 6.12 *Database* Gerai 3**

### 6.2.4 Transaksi dari gerai 4

Transaksi akan dilakukan dari *server* gerai ke *server* kereta dengan asumsi tiket telah dipesan dan telah tercatat dalam *database* pemesanan pada *server* gerai, kemudian saat transaksi *server* gerai akan meminta kode bayar dan akan dicocokan dengan data pemesanan. Jika kode bayar ditemukan maka *server* gerai akan meminta konfirmasi untuk melanjutkan transaksi, saat transaksi dilanjutkan *server* gerai akan mengirimkan *digest* data transaksi dan data transaksi ke *server* kereta untuk disimpan dalam *database server* kereta dan *database* gerai sendiri dan diganti statusnya menjadi “Sudah Dibayar”, rangkaian transaksi tersebut telah penulis lakukan dan dapat dilihat pada Gambar 6.13 .




**Gambar 6.13 Transaksi Dari Gerai 4**

Sehingga akan menghasilkan cacatan transaksi dengan tambahan *digest* data transaksi pada *server* gerai sebagai berikut:

| nomer | nama | id | tipe | tempatduduk | nomertiket | kereta | kelas | berangkat | tiba | statusbayar | harga | kodebayar | waktubayar | gerai | hashsebelum | hash |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | seseorang melalui gerai 3 | 45546345345223 | DEWASA | 21 B | 564342346723765 | PASUNDAN | EKONOMI | Gubeng 06.00 | Kiaracondong 23.00 | Belum Dibayar | 90000 | KODEBAYARGERAI3 | 2018-05-18 03:27:13.508579 | Gerai-03 | 0 | 87db152ff26a8b8c6f51372d1f8df534adf7bb7f |

**Gambar 6.14 *Database* Gerai 4**

### 6.2.5 Hasil Pencatatan Transaksi *Server* Kereta

Setelah melakukan beberapa transaksi dari empat gerai, data transaksi akan disimpan kedalam *database server* kereta sesuai urutan transaksi dan *digest* transaksi sebelumnya akan dicatatkan pada transaksi setelahnya sebagai tambahan data untuk dihitung *digest* transaksi tersebut. Catatan transaksi pada *database* kereta pada percobaan ini dapat dilihat pada Gambar 6.15 .

| nomer | nama | id | tipe | tempatduduk | nomertiket | kereta | kelas | berangkat | tiba | statusbayar | harga | kodebayar | waktubayar | gerai | hashsebelum | hash |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | seseorang melalui gerai 1 | 32454345345223 | DEWASA | 18 C | 4634533232354 | LOGAWA | EKONOMI | Bangil 09.00 | Lempuyangan 16.00 | Sudah Dibayar | 74000 | KODEBAYARGERAI1 | 2018-05-18 03:22:42.860913 | Gerai-01 | 0 | a77b9134abd93bfd3ce900aaa207348c204d0874 |
| 2 | seseorang melalui gerai 2 | 2654654634345345223 | DEWASA | 11 A | 34355434234567237765 | JAYABAYA | EKONOMI | Gubeng 21.00 | Tegal 04.00 | Sudah Dibayar | 120000 | KODEBAYARGERAI2 | 2018-05-18 03:25:57.874419 | Gerai-02 | a77b9134abd93bfd3ce900aaa207348c204d0874 | 690f1812d5abe28f1b9d6932ac963bdee37f3939 |
| 3 | seseorang melalui gerai 3 | 45546345345223 | DEWASA | 21 B | 5643423467237765 | PASUNDAN | EKONOMI | Gubeng 06.00 | Kiaracondong 23.00 | Sudah Dibayar | 90000 | KODEBAYARGERAI3 | 2018-05-18 03:27:13.508579 | Gerai-03 | 690f1812d5abe28f1b9d6932ac963bdee37f3939 | 2cdecf6c9d5f4e2207386505c2eaff98172ffd53 |
| 4 | Agung Pambudi | 2343523534234 | DEWASA | 12 B | 234234322343 2 | DOHO | EKONOMI | Surabaya 17.00 | Kediri 20.00 | Sudah Dibayar | 10000 | KODEBAYARGERAI4 | 2018-05-18 03:28:15.730142 | Gerai-04 | 2cdecf6c9d5f4e2207386505c2eaff98172ffd53 | 355c378f6bf2404df3228d0858ed2cab261dbd93 |

**Gambar 6.15 *Database Server* Kereta**

## 6.3 Pengujian Validitas Data Transaksi

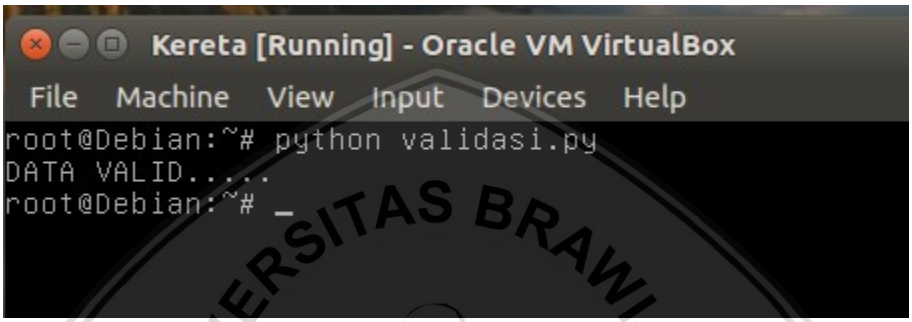
Pengujian validitas bertujuan untuk menguji ketepatan urutan data transaksi setiap barisnya yang telah tersimpan dalam *database* untuk memastikan integritas data transaksi sesuai tujuan dari sistem. Dalam pengujian ini penulis membuat sebuah program tambahan yang akan menghitung *digest* dari data transaksi dan *digest* data transaksi sebelumnya untuk membandingkan dengan *digest* data transaksi dan *digest* data transaksi sebelumnya yang telah tercatat dalam *database* mulai dari data baris pertama hingga data baris terakhir untuk memastikan data valid atau data tidak mengalami modifikasi.

Program ini dibuat dengan bahasa pemrograman Python, cara kerjanya pertama-tama akan melakukan *import library* PyMySQL untuk mengelola *database* dan modul *SHA-1* untuk melakukan perhitungan *digest,* kemudian inisialisasi variabel pembantu seperti *counter*, *temphash* dan maxno dilanjutkan dengan inisialisasi *database*. Setelah selesai inisialisasi maka program akan melihat nomor terbesar pada catatan data transaksi sebagai acuan perhitungan, program akan membandingkan perhitungan *hash* dengan catatan pada *database* dan nilai *digest* catatan *digest hash* sebelumnya yang ada pada catatan dengan nilai *digest hash* sebelumnya, jika selesai pada satu baris maka variabel *counter* akan ditambah sehingga program akan memeriksa baris selanjutnya hingga nilai variabel *counter* sama dengan banyaknya baris plus satu, hal ini menandakan bahwa semua baris telah diperiksa dan memenuhi syarat sehingga variabel *counter* mampu bertambah hingga melewati nilai maksimal baris catatan kemudian akan dicetak pesan bahwa data transaksi yang telah diperiksa valid, sebaliknya jika dalam pemeriksaan ditemukan data transaksi yang tidak memenuhi syarat maka program akan berhenti memeriksa dan mencetak pesan data transaksi tidak valid.

| Algoritme 5 : Fungsi Validitas | |
|---|---|
| 1 | `import pymysql` |
| 2 | `from sha1 import GenerateDigest` |
| 3 | |
| 4 | `counter=1` |

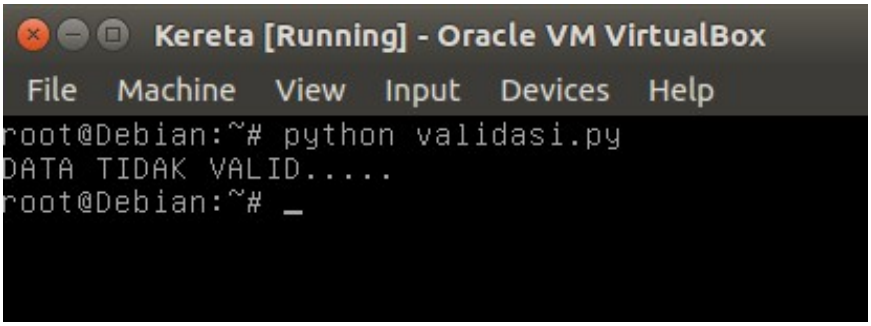
```
5   temphash="0"
6   maxno=1
7   db = pymysql.connect("localhost","root","br89q","transaksi" )
8   cursor = db.cursor()
9
10  sql = "SELECT max(nomor) FROM pembayaran;"
11  cursor.execute(sql)
12  b=cursor.fetchall()
13  for row in b :
14      nomor = (row[0])
15
16  if nomor != None :
17      maxno=int(nomor)
18      while True:
19          sql = "SELECT * FROM pembayaran WHERE nomor ='%s';"
20          cursor.execute(sql,(counter))
21          a=cursor.fetchall()
22          for row in a :
23
24  nomor,nama,id,tipe,tempatduduk,nomortiket,kereta,kelas,berangkat
    ,tiba,statusbayar,harga,kodebayar,waktubayar,gerai,hashsebelum,h
    ashsekarang=(row[0]),(row[1]),(row[2]),(row[3]),(row[4]),(row[5]
    ),(row[6]),(row[7]),(row[8]),(row[9]),(row[10]),(row[11]),(row[1
    2]),(row[13]),(row[14]),(row[15]),(row[16])
25
26
27  hashtemp=GenerateDigest(nomor+nama+id+tipe+tempatduduk+nomortike
    t+kereta+kelas+berangkat+tiba+statusbayar+harga+kodebayar+waktub
    ayar+gerai+hashsebelum)

28          if (hashtemp==hashsekarang and hashsebelum == temphash
29  and counter != (maxno+1)):
                temphash=hashsekarang
30              counter+=1
31
32          elif (counter==(maxno+1)):
33              print "DATA VALID....."
34
35          else:
36              print "DATA TIDAK VALID....."
37              break
38
39
40  elif nomor == None:
41      print "DATABASE KOSONG....."
42
43  db.close()
44
```

Pada pengujian fungsionalitas sebelumnya penulis melakukan percobaan terhadap fungsionalitas sistem dan menghasilkan catatan data transaksi pada *database server* kereta, maka pada pengujian kali ini penulis akan menjalankan program yang telah dibuat untuk mengoreksi catatan data transaksi pada pengujian fungsionalitas sebelumnya. Program dijalankan pada *server* kereta serta dikoneksikan dengan *database server* kereta, setelah program dijalankan didapati bahwa catatan data transaksi yang ada pada *database server* kereta valid dan integritasnya terjamin seperti yang bisa dilihat pada Gambar 6.16.




**Gambar 6.16 Pengujian Validitas Data Transaksi 1**

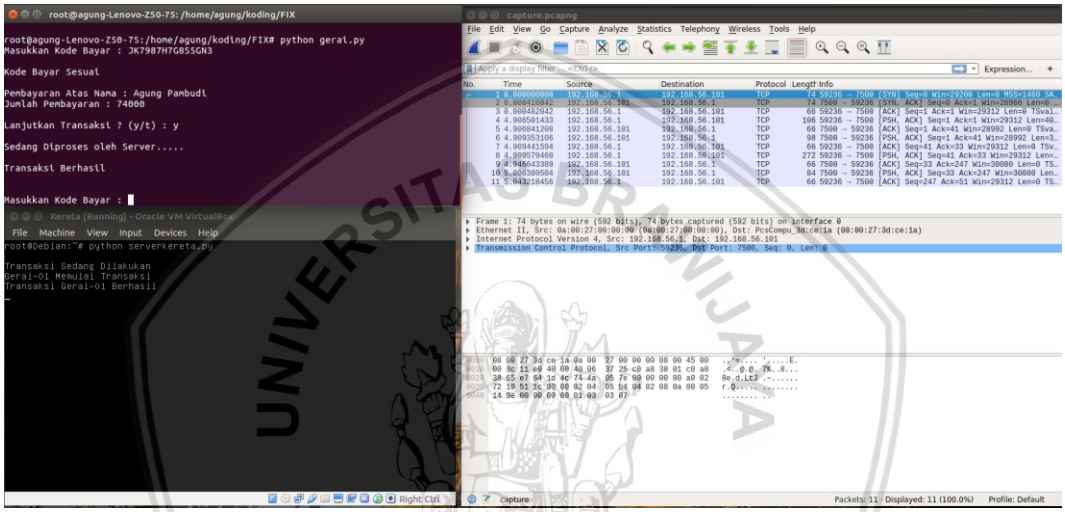
Jika pada pengujian diatas dapat dibuktikan bahwa sistem akan menghasilkan catatan data yang valid, maka selanjutnya penulis akan mencoba membuktikan jika komponen data pada catatan data transaksi ada yang dirubah akan merusak susunan *hashing* sehingga integritas data tidak dapat dipercaya. Penulis akan merubah komponen data yang dihasilkan oleh percobaan sebelumnya dan akan menjalankan program validitas ulang, sehingga didapatkan hasil seperti Gambar 6.17 dan dapat disimpulkan bahwa dengan skema ini jika komponen data transaksi berubah akan menghasilkan *digest* yang berbeda sehingga akan memudahkan ketika membuktikan integritas catatan data transaksi.




**Gambar 6.17 Pengujian Validitas Data Transaksi 2**

## 6.4 Pengujian *Quality of Services*

Setelah beberapa pengujian yang telah penulis lakukan dan didapatkan hasil bahwa sistem dapat berjalan dengan baik dan benar, pada pengujian ini penulis akan melakukan beberapa pengujian di antaranya berapa waktu eksekusi sistem, berapa *throughput* koneksi antara *server* gerai dan *server* kereta, adakah *packet loss* dalam transmisi data antara *server* gerai dan *server* kereta. Penulis menggunakan aplikasi Wireshark untuk mengamati lalu lintas pada jaringan yang menghubungkan antara *server* gerai dan *server* kereta seperti pada Gambar 6.18, dan hasil pengujian ini adalah sebagai berikut:




**Gambar 6.18 Pengujian Validitas Data Transaksi**

### 6.4.1 Waktu Eksekusi Transaksi

Waktu yang dibutuhkan dalam sekali transaksi adalah dengan menghitung waktu saat paket pertama dikirimkan sampai paket terakhir diterima, dari tangkapan aplikasi Wireshark dapat diketahui bahwa paket pertama dikirim pada pukul 00:10:49 dan paket terakhir yang merupakan balasan dari *server* kereta diterima pada pukul 00:10:54 sehingga lama waktu sekali transaksi adalah lima detik seperti yang dapat dilihat pada Gambar 6.19


**Gambar 6.19 Waktu Eksekusi Transaksi**

### 6.4.2 *Throughput*

*Throughput* adalah kemampuan sebuah jaringan mentransmisikan beberapa data dalam satu waktu tertentu atau sering juga disebut sebagai *bandwidth* aktual, cara menghitungnya ialah mebangi banyaknya data yang berhasil di transmisikan dengan seberapa lama data tersebut di transmisikan, terlihat dari Gambar 6.20 banyaknya data yang di transmisikan adalah 1038 *byte*s dan waktu yang dibutuhkan untuk mentransmisikan adalah 5,043 detik dari data tersebut dapat diketahui bahwa *throughput* dari transaksi antara *server* gerai dan *server* kereta adalah 205,829 *byte*s/detik.

| | |
|---|---|
| Time span, s | 5.043 |
| Average pps | 2.2 |
| Average packet size, B | 94.5 |
| Bytes | 1038 |
| Average bytes/s | 205 |

**Gambar 6.20 *Throughput***

### 6.4.3 *Packet Loss*

Sistem ini menggunakan *protocol tcp* untuk mentransmisikan data, secara teori jika koneksi tidak terputus maka tidak akan mungkin ada *packet loss*, karena sifat *protocol tcp* yang akan melakukan transmisi ulang jika ditemukan adanya *packet* yang hilang. Dengan percobaan ini penulis membuktikan bahwa dalam transmisinya sistem ini tidak menghasilkan *packet loss* seperti yang dapat dilihat pada Gambar 6.21 bahwa *packet* yang berhasil di transmisikan adalah 100%.

| Measurement | Captured | Displayed |
|---|---|---|
| Packets | 11 | 11 (100.0%) |

**Gambar 6.21 Transmisi *Packet***

# BAB 7 KESIMPULAN DAN SARAN

## 7.1 Kesimpulan

Setelah melakukan penelitian yang berjudul Perancangan Sistem Pengamanan Data Transaksi Pada *Database* Terdistribusi Menggunakan Metode *Hashing* dengan beberapa tahapannya, penulis dapat menyimpulkan beberapa hal sebagai berikut:

1. Algoritme *Secure Hash Algorithm* 1 (*SHA-1*) adalah algoritme untuk menjaga integritas yang pernah dijadikan standar algoritme *hashing* dan memiliki keamanan yang cukup baik karena arsitekturnya memiliki ukuran 512 blok dan menghasilkan 160 bit *digest*. Hal ini menjadikan *SHA-1* tidak mudah untuk diserang dengan metode *brute force* karena akan memerlukan waktu yang lama.
2. Dalam sistem ini penulis telah melakukan implementasi algoritme *Secure Hash Algorithm* 1 (*SHA-1*) dengan bahasa pemrograman yang sama dengan sub-program yang lain yaitu bahasa pemrograman Python dan dapat disimpulkan bahwa sub-program *SHA-1* dapat berjalan dengan baik dan benar sebagaimana fungsinya.
3. Dari percobaan yang telah penulis lakukan dapat disimpulkan bahwa catatan data transaksi yang telah diamankan oleh sistem ini tidak dapat dimanipulasi, karena satu catatan saling terkait dengan catatan yang lain pada *database* sehingga apabila satu catatan diubah maka akan merusak catatan yang lain, hal ini dapat diketahui menggunakan program penguji validitas data transaksi yang telah penulis buat sebelumnya.

## 7.2 Saran

Penulis menyadari masih banyak kekurangan dalam penelitian ini, di antaranya adalah tidak amannya data transaksi yang ditransmisikan dari serangan *man in middle attack* dikarenakan dalam penelitian ini penulis hanya berfokus pada integritas data serta saat ini penggunaan algoritme *hashing Secure Hash Algorithm* 1 (*SHA-1*) mulai digantikan dengan algoritme-algoritme yang lebih baru guna meningkatkan keamanan dan kecepatan perhitungan. Hal ini dapat dimanfaatkan sebagai topik penelitian untuk menyempurnakan penelitian yang telah penulis lakukan, penulis menyarankan penggunaan algoritme *Secure Hash Algorithm* 3 (*SHA*-3) sebagai pengganti algoritme *SHA-1* dan penggunaan algoritme enkripsi untuk mengamankan data transaksi pada saat ditransmisikan guna menghindari adanya serangan *man in middle attack*.

# DAFTAR PUSTAKA


Dang, Q. H., 2015. *Secure Hash Standard (SHS)*. Federal Information Processing Standards Publication.

Menezes, A. J., Katz, J., van Oorschot, P. C. & Vanstone, S. A., 1996. *Handbook of Applied Cryptography*. s.l.:CRC Press.

Nakamoto, S., 2008. *Bitcoin: A Peer-to-Peer Electronic Cash System*.

Pamungkas A. A., 2006. *Implementasi Algoritma MD5, SHA1, dan RC4 untuk Sistem Kriptografi Pada Aplikasi Mobile Internet Berbasis JAVA*.

Dewanagn, R. R., Katz, J., Thombre, D. & Sharma, K., 2015. *Implementation of Secure Hash Algorithm Using JAVA Programming*.